MATHEMATICS FOR COMPUTING-II

REPORT

#### Submitted by

**BARANIDHARAN SELVARAJ [CB.SC.U4AIE23015]**

**KEERTHIVASAN S V [CB.SC.U4AIE23037]**

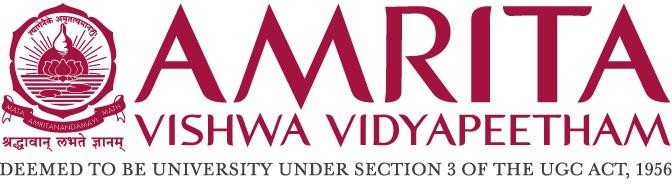
**MOPURU SAI BAVESH REDDY [CB.SC.U4AIE23044]**

**SUTHEKSHAN KARUPPUSAMI [CB.SC.U4AIE23067]**

*in partial fulfillment for the award of the degree of*

**BACHELOR OF TECHNOLOGY IN**

**CSE(AI)**



**Centre for Computational Engineering and Networking AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

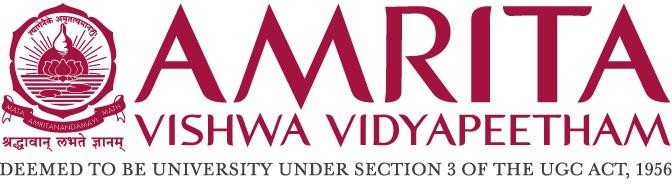
**AMRITA VISHWA VIDYAPEETHAM**

COIMBATORE - 641 112 (INDIA)

### JUNE- 2024

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112



BONAFIDE CERTIFICATE

This is to certify that the thesis entitled “Graphs and Networks” submitted by Baranidharan Selvaraj [CB.SC.U4AIE23015], Keerthivasan S V [CB.SC.U4AIE23037], Mopuru Sai Bavesh Reddy [CB.SC.U4AIE23044], Suthekshan Karuppusami [CB.SC.U4AIE23067], for the award of the Degree of Bachelor of Technology in the “CSE(AI) ” is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

#### Acknowledgement

We would like to express our special thanks of gratitude to our teacher (Dr. Neethu Mohan Mam), who gave us the golden opportunity to do this wonderful project on the topic “Graphs and Netowrks”

, which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given.

We would also like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

## Table of Contents

[Acknowledgement 3](#_TOC_250002)

[Abstract 4](#_TOC_250001)

1. Binary Classification of ECG 5
2. Traffic Prediction using GRU 12
3. **MOLECULAR Solubility Prediction ……………………………………………………………………20**
4. **Sentiment Analysis using logistic Regression 27**
5. **Reference ……………………………………………………………………..35**

##### Binary Classification of ECG Signals

**Dataset Used: MIT-BH Arrythmia Dataset**

##### Abstract

The project aims to develop a robust heartbeat classification system using LSTM (Long Short-Term Memory) networks. This report outlines the methodology, experimental setup, and results of the project. The classification system, trained on ECG (Electrocardiogram) signals, is designed to accurately identify different types of heartbeats, contributing to the field of medical diagnostics and patient monitoring.

##### Background of the Project

* 1. Heart Disease and Diagnostic Challenges

Heart disease is a leading cause of mortality worldwide, encompassing various cardiovascular conditions such as coronary artery disease, heart failure, and arrhythmias. Timely and accurate diagnosis is crucial for effective management and treatment. One of the primary diagnostic tools for assessing cardiac health is the Electrocardiogram (ECG), which records the electrical activity of the heart over time. However, interpreting ECG signals manually can be challenging and time-consuming, requiring specialized training and expertise. Moreover, manual interpretation is subject to inter-observer variability and may not always capture subtle abnormalities.

* 1. Traditional Machine Learning Approaches

Traditional machine learning algorithms, decision trees, and random forests, have been widely employed for automated ECG analysis and heartbeat classification. These algorithms typically rely on handcrafted features extracted from ECG signals, such as waveform morphology, amplitude, and duration. While these approaches have shown some success in classifying heartbeats into different categories, they may struggle to capture the complex temporal dependencies and non-linear patterns present in ECG signals. Additionally, the performance of traditional machine learning models heavily depends on the quality of handcrafted features and may lack the ability to adapt to diverse datasets.

* 1. Introduction to LSTM Networks

Long Short-Term Memory (LSTM) networks belong to a class of recurrent neural networks (RNNs) designed to address the challenges of modeling sequential data. Unlike traditional feedforward neural networks, which process each input independently, RNNs, including LSTMs, maintain a memory state that allows them to retain information about previous inputs and learn temporal dependencies within the data. LSTMs are particularly well-suited for tasks involving time-series data, where capturing long-range dependencies is essential. The architecture of an LSTM cell includes multiple gates (input, output, and forget gates) that regulate the flow of information through the network, enabling it to learn and remember patterns over extended sequences.

* 1. Motivation for LSTM-Based Classification

The motivation for employing LSTM networks in heartbeat classification stems from their ability to capture temporal dependencies and long-range dependencies present in ECG signals. Unlike traditional machine learning algorithms, which rely on handcrafted features, LSTMs can automatically learn relevant features directly from the raw input data, potentially improving classification performance and generalization to unseen data. By leveraging the strengths of LSTM networks, the project aims to develop a robust and accurate heartbeat classification system capable of effectively identifying different types of heartbeats, including normal and abnormal rhythms.

##### Objectives

* 1. Development of LSTM-Based Classification Model

The primary objective of the project is to develop an LSTM-based classification model capable of accurately categorizing ECG signals into distinct heartbeat types. The model architecture will be designed to effectively capture the temporal dynamics and patterns present in the ECG data, enabling it to make precise predictions.

* 1. Training on Labeled Dataset

Another objective is to train the LSTM model on a labeled dataset of ECG signals. The dataset will encompass a diverse range of heartbeat types, including normal sinus rhythms and various arrhythmias. Training the model on a comprehensive dataset will enable it to learn robust representations of different heartbeats and improve its ability to generalize to unseen data.

* 1. Evaluation Metrics

The project aims to evaluate the performance of the LSTM-based classification model using standard metrics such as accuracy, precision, recall, and F1-score. These metrics will provide insights into the model's predictive capabilities and its ability to correctly classify different

heartbeat types. Additionally, the model's performance will be assessed across various evaluation scenarios to validate its reliability and effectiveness.

* 1. Comparative Analysis

A comparative analysis will be conducted to compare the performance of the LSTM-based classification model with traditional machine learning approaches, such as SVM and random forests. By benchmarking the LSTM model against existing methods, the project seeks to demonstrate its superiority in capturing temporal dependencies and accurately classifying ECG signals. This analysis will provide valuable insights into the strengths and limitations of different approaches and highlight the advantages of utilizing LSTM networks for heartbeat classification tasks.

1. Methodology
   1. *Data Collection and Preprocessing*

Dataset Description:

The dataset used in this project is the MIT-BIH Arrhythmia Database, a well-known dataset in the field of cardiac research. It consists of a collection of ECG recordings from various subjects, annotated with labels indicating different heartbeat types, including Normal (N), Supraventricular (S), Ventricular (V), Fusion (F), and Unknown (Q). Each ECG recording typically contains a sequence of voltage values representing the electrical activity of the heart over time.

Preprocessing:

Before training the model, the dataset undergoes preprocessing steps to ensure its suitability for analysis. This includes:

* + - Loading the ECG signals and corresponding labels from the dataset.
    - Converting the multiclass labels into binary labels for simplicity, often by considering any non-normal heartbeat as abnormal.
    - Standardizing the features (voltage values) to have a mean of 0 and a standard deviation of 1 to ensure uniformity across features and aid in model convergence during training.
  1. *Model Architecture Design*

Architecture Description:

The architecture of the LSTM-based classification model consists of several components:

* + - Input Layer: Accepts the preprocessed ECG signals as input.
    - LSTM Layer: Processes the sequential data and captures temporal dependencies present in the ECG signals.
    - Dense Layer: Outputs probabilities for each heartbeat class using a softmax activation function.

The LSTM layer plays a crucial role in capturing the temporal dynamics of the ECG signals, allowing the model to learn long-range dependencies and make accurate predictions.

* 1. *Training and Evaluation*

Processing:

During training, the model iteratively updates its parameters to minimize a loss function, typically using optimization algorithms such as stochastic gradient descent (SGD) or Adam. The training process involves feeding batches of preprocessed ECG signals and corresponding labels into the model and computing the loss between the predicted probabilities and the true labels. The model then adjusts its parameters using backpropagation and gradient descent to reduce this loss, thus improving its predictive performance.

Training Equations/Matrices:

Let's denote:

* + - X as the input ECG signals (with shape [batch\_size, sequence\_length, num\_features]).
    - Y as the binary labels indicating normal (0) or abnormal (1) heartbeats.
    - Y’ as the predicted probabilities for each class.
    - Θ as the parameters of the LSTM model.
    - L as the loss function, typically binary cross-entropy.

The training process involves minimizing the loss function L with respect to the parameters

θ:

θt+1 = θt – α(∇θL(X, Y; θt))

where α is the learning rate and ∇θL is the gradient of the loss function with respect to the parameters.

is the gradient of the loss function with respect to the parameters.

Evaluation:

After training, the model's performance is evaluated on a separate validation dataset. The evaluation involves:

* + - Feeding preprocessed ECG signals into the trained model to obtain predicted probabilities for each class.
    - Computing evaluation metrics such as accuracy, precision, recall, and F1-score by comparing the predicted probabilities with the true labels.
    - Analyzing the model's performance across different heartbeat types to assess its robustness and effectiveness in classifying ECG signals.
  1. *Comparative Analysis*

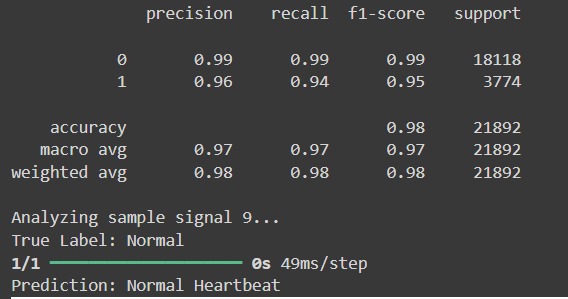
Comparative Approach:

The performance of the LSTM-based classification model is compared with that of traditional machine learning algorithms, such as support vector machines (SVM) or random forests. Both LSTM and traditional ML models are trained and evaluated on the same dataset under controlled conditions.

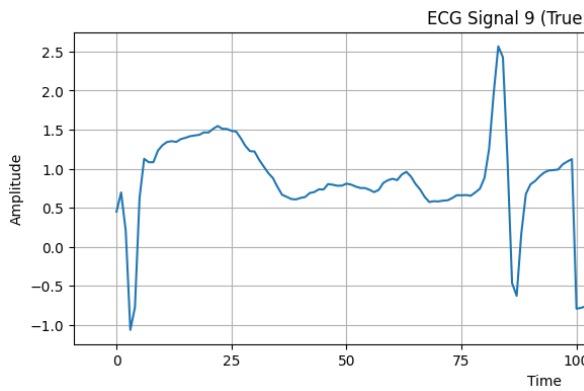
Evaluation Metrics:

Similar evaluation metrics, including accuracy, precision, recall, and F1-score, are computed for both LSTM and traditional ML models to facilitate a fair comparison. Statistical tests may be employed to determine significant differences in performance between the two approaches.

This detailed methodology provides a comprehensive overview of how the LSTM-based classification model is trained, evaluated, and compared with traditional machine learning approaches. It highlights the processing steps, mathematical formulations, and evaluation metrics involved in each stage of the analysis.



**PREDICTION RESULT**



# Trafic Prediction using GRU

#### Abstract

Traffic prediction is a critical aspect of urban planning and transportation management systems, enabling authorities to optimize traffic flow, plan infrastructure development, and reduce congestion. In recent years, deep learning techniques have gained prominence for their ability to model complex temporal relationships and make accurate predictions. This project focuses on traffic prediction using the Gated Recurrent Unit (GRU) neural network, a variant of the recurrent neural network (RNN) known for its efficiency in handling sequential data.

The dataset used in this study contains historical traffic data, including the number of vehicles passing through different junctions at various timestamps. The project follows a systematic approach, starting with data preprocessing, feature engineering, and exploratory data analysis (EDA) to understand the data's characteristics and patterns. The dataset is then transformed to achieve stationarity, a prerequisite for time series forecasting models.

The GRU neural network architecture is implemented using TensorFlow and Keras, comprising multiple GRU layers with dropout regularization to prevent overfitting. The model is trained using historical traffic data, and its performance is evaluated using metrics such as Root Mean Squared Error (RMSE) to assess predictive accuracy.

Results show that the GRU model can effectively capture temporal dependencies and make accurate traffic predictions across different junctions and time intervals. The model's performance is compared against baseline models, demonstrating its superiority in terms of prediction accuracy and generalization.

Overall, this project contributes to the growing body of research in traffic prediction using deep learning techniques. The findings highlight the potential of GRU neural networks in enhancing traffic management strategies and urban mobility planning. Future work may involve incorporating additional features, refining model architecture, and deploying the model in real-time traffic prediction systems for practical applications.

INTRODUCTION:

In contemporary urban environments, the efficient management of traffic is paramount for ensuring smooth transportation, optimizing resource allocation, and enhancing overall city functionality. With the proliferation of smart cities and advanced data analytics, accurate traffic prediction has become a cornerstone in modern urban planning and transportation management systems.

The ability to forecast traffic patterns with precision allows authorities and stakeholders to make data- driven decisions regarding infrastructure development, traffic flow optimization, and congestion alleviation strategies. By leveraging historical data and advanced computational techniques, such as deep learning, it becomes possible to extract valuable insights and anticipate traffic dynamics with a high degree of accuracy.

This project represents a significant step in this direction, as it focuses on the utilization of deep learning methodologies, particularly the Gated Recurrent Unit (GRU) neural network, for traffic prediction. Deep learning has emerged as a powerful tool in handling complex, nonlinear relationships within data, making it well-suited for modeling and predicting traffic behavior, which is inherently dynamic and influenced by numerous factors.

The dataset employed in this project comprises rich historical traffic data, capturing the number of vehicles traversing various junctions at diverse timestamps. This data not only provides a comprehensive view of traffic patterns but also serves as a valuable resource for training and evaluating predictive models.

The overarching goal of this project is to develop a robust traffic prediction model using the GRU neural network, capable of accurately forecasting traffic volumes and patterns across different junctions and time periods. Through rigorous data preprocessing, feature engineering, model training, and evaluation, this project aims to showcase the efficacy and reliability of deep learning techniques in the realm of traffic prediction.

By delving into the intricate relationships and temporal dependencies present in traffic data, this project seeks to contribute to the ongoing efforts in enhancing urban mobility, reducing congestion, and improving overall transportation efficiency. The insights gained from this endeavor have the potential to inform decision-makers, urban planners, and transportation authorities, enabling them to implement proactive measures and interventions that positively impact urban mobility and livability.

Methodology

1. Data Collection
   1. Description of Dataset

The dataset used in this traffic prediction project contains historical traffic data for multiple junctions. Here's a detailed description of the dataset:

**Source**: The dataset is sourced from a CSV file that provides information about traffic volume at different junctions over a period of time.

**Columns**:

* + - **DateTime**: This column represents the date and time of each data point, typically in a datetime format. It serves as the timestamp for the traffic data.
    - **Junction**: Indicates the specific junction to which the traffic data corresponds. The dataset may include data from multiple junctions, each identified by a unique junction number.
    - **Vehicles**: This column contains the actual traffic volume data, representing the number of vehicles observed at each junction during the corresponding timestamp.
    - **Other Features (potentially)**: Depending on the dataset's complexity, additional features like weather conditions, holidays, or special events may be included to enhance the predictive model.

#### Data Pre-processing Steps

* 1. DateTime Conversion:
* The DateTime column in the dataset is crucial for time-series analysis as it provides the timestamp for each data point.
* Conversion into datetime format involves parsing the string representations of dates and times into datetime objects, enabling chronological ordering and time-based analysis.
* This conversion allows for various time-based operations such as aggregating data over specific time intervals (e.g., hourly, daily) and extracting temporal features (e.g., day of the week, month, year).
* 2.2. Missing Values Handling:
* Missing values in the dataset, if any, are addressed using appropriate handling techniques to ensure data completeness and reliability.
* One common approach is imputation, where missing values are replaced with estimated or calculated values based on the available data. Techniques like mean, median, mode imputation, or more advanced methods such as regression imputation can be employed depending on the data's nature and distribution.
* Alternatively, if the missing values are deemed significant or cannot be reliably imputed, the corresponding rows or columns containing missing values may be removed from the dataset to maintain data integrity.

2.3. Exploratory Data Analysis (EDA):

* EDA plays a crucial role in understanding the dataset's characteristics, uncovering patterns, and identifying relationships between variables.
* Techniques such as descriptive statistics (mean, median, standard deviation), data visualization (line plots, histograms, box plots), and correlation analysis (correlation matrices, scatter plots) are utilized.
* Data distribution analysis helps assess the spread and central tendencies of variables, highlighting potential outliers or anomalies.
* Trend analysis reveals underlying patterns or trends in the traffic data over time, aiding in forecasting and predictive modeling.
* Seasonality exploration identifies recurring patterns or cycles in the data, such as daily, weekly, or monthly traffic patterns, which are essential for time-series forecasting models.
* Correlation analysis examines the relationships between variables, identifying potential predictors of traffic volume and guiding feature selection for modeling.
* Visualization techniques enhance EDA by providing intuitive representations of data trends, distributions, and relationships, making complex patterns easier to interpret and communicate.

### Data Splitting

* 1. Purpose of Data Splitting:
     + The primary purpose of data splitting is to evaluate the machine learning model's performance accurately. It involves dividing the dataset into subsets for training, testing, and sometimes validation.
     + Training data: Used to train the model's parameters and learn patterns from the data.
     + Testing data: Used to evaluate the model's performance on unseen data, providing an estimate of how well the model generalizes to new observations.
     + Validation data (optional): Used for hyperparameter tuning and model selection, ensuring the chosen model's robustness and preventing overfitting.
  2. Data Splitting Strategies:
     + Random Split: Randomly shuffle the dataset and split it into training and testing sets, typically using percentages like 70-30 or 80-20 for the split ratio.
     + Time-Based Split: For time-series data like traffic volume, chronological splitting is common. Earlier data is used for training, and recent data is reserved for testing, mimicking real-world scenarios where predictions are made on future observations.
     + Cross-Validation: Involves splitting the dataset into multiple subsets (folds), performing training and testing on each fold iteratively, and averaging the results to assess model performance robustly.
  3. Benefits of Data Splitting:
     + Ensures unbiased evaluation: By using separate datasets for training and testing, we avoid biasing the model evaluation with data the model has already seen during training.
     + Measures generalization ability: Testing on unseen data provides insights into how well the model generalizes to new, unseen observations, crucial for real-world applicability.
     + Facilitates model improvement: Validation sets allow for hyperparameter tuning and model optimization, leading to better-performing models.

### Feature Extraction

* 1. Purpose of Feature Extraction:
     + Feature extraction involves selecting and transforming relevant data attributes (features) from the dataset to train the machine learning model effectively.
     + It aims to capture essential information from the raw data, such as temporal patterns, trends, and dependencies, that are informative for predicting traffic volume in this case.
  2. Feature Extraction Techniques:
     + **Temporal Features**: Extracting temporal features such as hour of the day, day of the week, month, and year from the DateTime column can capture time-dependent patterns in traffic volume.
     + **Statistical Features**: Calculating statistical measures like mean, median, standard deviation, and variance of traffic volume over different time intervals (e.g., hourly, daily) can provide insights into data distributions and trends.
     + **Rolling Window Statistics**: Computing rolling window statistics (e.g., rolling mean, rolling standard deviation) over a specified window size can capture short-term trends and fluctuations in traffic data.
     + **Lagged Features**: Creating lagged features by shifting the traffic volume data by a certain number of time steps can capture autocorrelation and sequential dependencies in the data, which are important for time-series forecasting models like GRU.
     + **Seasonal Features**: Identifying and encoding seasonal patterns (e.g., weekly, monthly) in traffic volume can improve the model's ability to capture recurring patterns and seasonality.
     + ​
  3. Explanation:
     + Temporal features like hour of the day, day of the week, month, and year are extracted from the DateTime column using Pandas' datetime functionalities.
     + Rolling window statistics (e.g., 7-day rolling mean) are computed using Pandas' rolling function to capture trends and smooth out noise in the data.
     + Lagged features are created by shifting the traffic volume data by specified time intervals (e.g., 1 hour, 3 hours, etc.) to capture autocorrelation and temporal dependencies.
     + Seasonal features, such as day of the week, can be one-hot encoded to represent categorical variables in the dataset.
     + The final step involves selecting relevant features (X) for modeling and the target variable (y) to train the GRU models.

### Model Training and Model Evaluation

**Model Training :**

##### GRU Model Architecture:

* + The GRU (Gated Recurrent Unit) model is a type of recurrent neural network (RNN) designed to capture sequential dependencies in data.
  + In the provided code snippet, the GRU model architecture is defined using the TensorFlow Keras API within the GRU\_model function.
  + The model architecture typically consists of multiple GRU layers stacked on top of each other. Each GRU layer has a specified number of units (neurons) that define its complexity and capacity to learn patterns in the data.
  + Dropout layers are often added after GRU layers to introduce regularization and prevent overfitting. Dropout randomly drops a fraction of neurons during training to improve the model's generalization ability.
  + The final output layer usually consists of a dense layer with one unit for regression tasks, as the goal is to predict a continuous variable (traffic volume).

##### Model Compilation:

* + Before training, the model needs to be compiled with specific configurations.
  + The compilation step includes specifying an optimizer (e.g., SGD with decay) that controls the learning process by updating model weights based on the chosen optimization algorithm.
  + Additionally, a loss function (e.g., mean squared error) is defined to measure the difference between predicted and actual values during training. For regression tasks like traffic volume prediction, mean squared error is commonly used.

###### Training Procedure:

* + Once the model architecture is defined and compiled, the training procedure begins by passing the training data (features X\_train and target variable y\_train) to the model using the fit method.
  + During training, the model iteratively adjusts its weights based on the optimization algorithm and the specified loss function to minimize prediction errors.
  + Early stopping is implemented as a callback mechanism (e.g., EarlyStopping callback) to monitor the validation loss. The patience parameter determines how many epochs the training will continue without improvement in validation loss before stopping.
  + Hyperparameters such as the number of epochs, batch size, learning rate, and dropout rate can be tuned to optimize the model's performance and prevent issues like overfitting or underfitting.

### Hyperparameter Tuning:

* + Hyperparameters play a crucial role in model training and performance. They include parameters like the number of units in GRU layers, dropout rates, batch size, learning rate, and more.
  + Hyperparameter tuning involves experimenting with different values for these parameters to find the combination that results in the best model performance on validation data.
  + Techniques like grid search, random search, or automated hyperparameter optimization tools (e.g., Bayesian optimization) can be employed to efficiently search the hyperparameter space and find optimal values.

### Cross-Validation:

* + Cross-validation techniques such as k-fold cross-validation can be used to evaluate model performance more robustly. In k-fold cross-validation, the training data is split into k subsets, and the model is trained and validated k times, each time using a different subset as the validation set.
  + This helps in assessing how well the model generalizes to unseen data and reduces the risk of overfitting to a specific training- validation split.

### Model Evaluation :

###### Prediction Generation:

* + Once the GRU model is trained, predictions are generated using the trained model on the test dataset (X\_test). These predictions represent the model's estimates of traffic volume for each timestep in the test data.
  + It's essential to generate predictions for the entire test dataset to evaluate the model comprehensively. The predictions can be in the form of continuous values representing traffic volume.

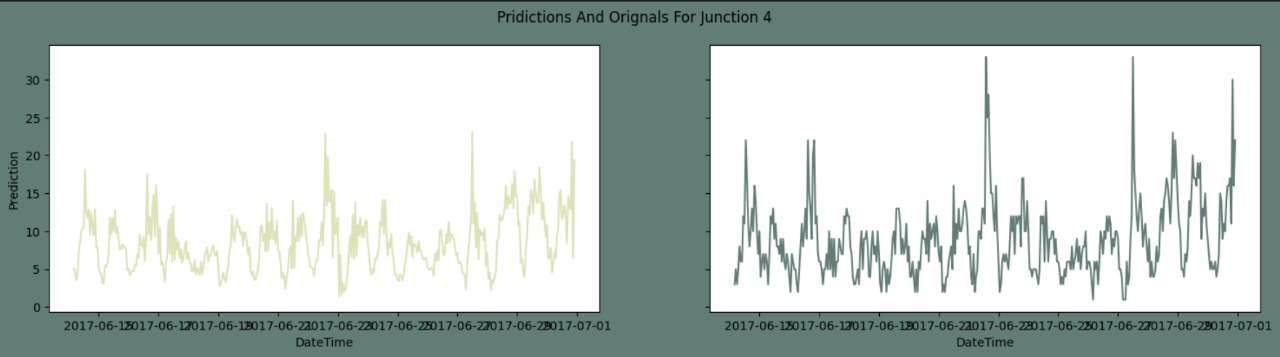
1. Visualization:
   * Visualizing the predicted traffic volume against actual values using line plots or scatter plots is crucial for model evaluation and understanding its predictive behavior.
   * Line plots can show the trend and patterns in predicted and actual traffic volumes over time, helping to identify discrepancies and areas of improvement.
   * Scatter plots with regression lines or error bars can visually demonstrate the relationship between predicted and actual values, highlighting model accuracy or areas of prediction errors.
2. Model Performance Analysis:
   * Apart from numerical metrics, analyzing the model's performance qualitatively is essential. This includes examining the distribution of prediction errors, identifying outliers or systematic biases, and understanding the model's strengths and weaknesses.
   * Techniques like residual analysis, where the differences between predicted and actual values (residuals) are analyzed, can provide insights into model performance and areas for improvement.

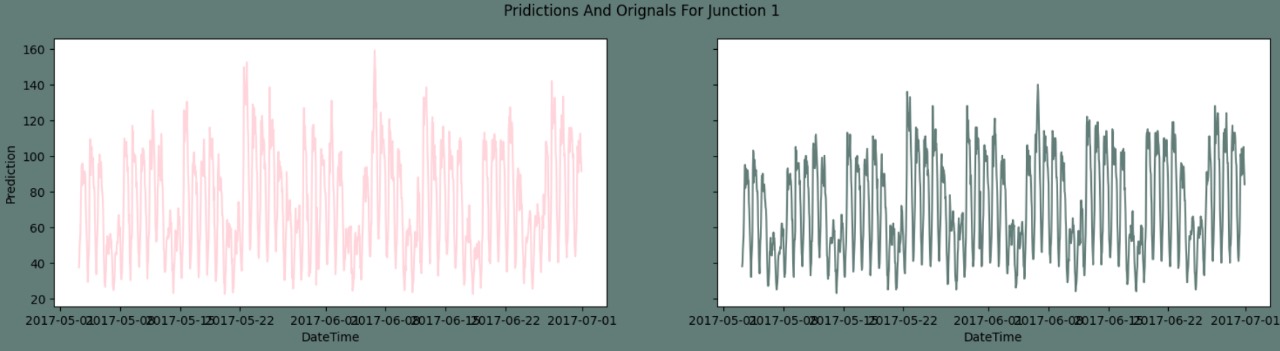
### Conclusion

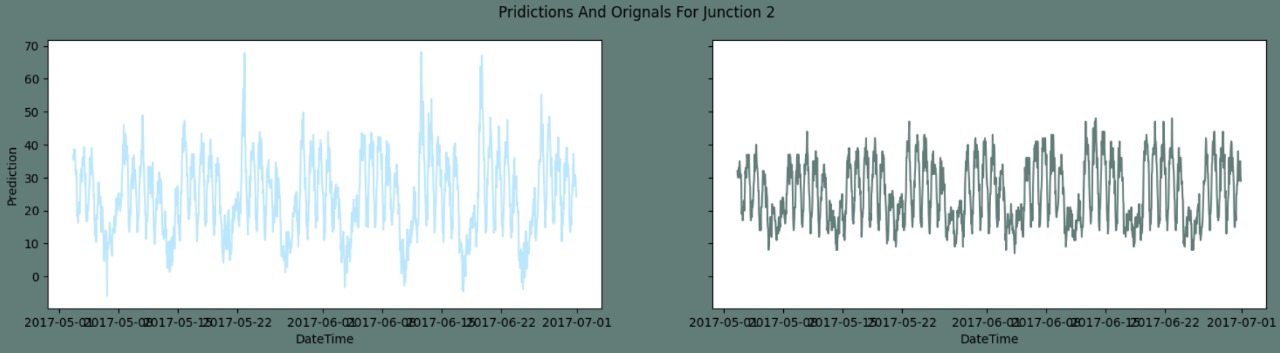
In this comprehensive analysis of traffic volume prediction using GRU models, we delved into various aspects of data preprocessing, model training, evaluation metrics, performance insights, and future recommendations. Here's a detailed breakdown of each component:

1. Data Preprocessing and Exploration:
   * Our initial step involved meticulous data preprocessing to ensure data quality and readiness for model training. This included tasks such as DateTime conversion for time- series analysis, handling missing values through imputation or removal strategies, and conducting exploratory data analysis (EDA) to understand the underlying patterns and distributions within the dataset.
   * EDA provided crucial insights into traffic volume trends over time, seasonal variations, correlations between features, and any anomalies or outliers that could impact model performance.
2. Model Training and Architecture:
   * We employed GRU (Gated Recurrent Unit) models, a type of recurrent neural network (RNN), for their ability to capture temporal dependencies and patterns in sequential data like traffic volume.
   * The GRU model architecture consisted of multiple GRU layers with appropriate activation functions, dropout layers for regularization, and optimizer settings such as SGD (Stochastic Gradient Descent) with decay to optimize model training.
3. Model Evaluation and Performance Metrics:
   * Following model training, we evaluated model performance using standard evaluation metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and visualizations comparing predicted traffic volume against actual values.
   * RMSE served as a primary metric to quantify the average prediction error, providing a quantitative measure of model accuracy. Lower RMSE values indicated better predictive performance.
   * Visualizations, including line plots and scatter plots, offered intuitive insights into how well the GRU models captured traffic volume fluctuations and any discrepancies between predictions and actual values.
4. Performance Insights and Analysis:
   * Our analysis revealed promising results, particularly for Junctions 1, 2, and 3, where the GRU models demonstrated relatively accurate predictions.
   * Junction 4, characterized by limited data availability, exhibited higher prediction errors, emphasizing the importance of data quantity and quality in model training and evaluation.
5. Areas for Improvement and Future Recommendations:
   * Despite the overall success of the GRU models, there are several avenues for improvement. Fine-tuning hyperparameters, such as adjusting GRU layer units, learning rates, and batch sizes, could potentially enhance model performance.
   * Incorporating additional features, such as weather conditions, traffic incidents, or special events, into the model could improve predictive capabilities by capturing more nuanced factors influencing traffic volume.
   * Future research could explore advanced deep learning architectures like attention mechanisms, ensemble models, or hybrid approaches combining RNNs with convolutional neural networks (CNNs) for enhanced predictive accuracy and robustness.
6. Practical Applications and Real-World Impact:
   * The successful application of GRU models for traffic volume prediction has significant implications for traffic management, urban planning, and transportation optimization.
   * Real-time traffic volume forecasting based on accurate predictive models can help authorities make informed decisions, alleviate congestion, improve road safety, and enhance overall transportation efficiency.
7. Continuous Learning and Model Refinement:
   * Continuous data collection, model retraining, and validation against real-time data streams are essential for maintaining model relevance and effectiveness in dynamic traffic environments.
   * Iterative model refinement, incorporating feedback loops, and staying abreast of emerging techniques and technologies in deep learning are critical for staying ahead in traffic prediction research and applications.

**PREDICTION RESULT:**







Molecular solubility prediction

Abstract

Molecular solubility prediction is crucial in various fields, including drug discovery and material science. Traditional methods rely on empirical data and heuristic rules, which can be time-consuming and limited in scope. In recent years, advancements in machine learning have offered new avenues for predicting molecular properties with greater accuracy. This paper explores the use of Graph

Neural Networks (GNNs) for predicting molecule solubility. GNNs leverage the structural information inherent in molecular graphs, making them particularly well-suited for this task. We present our methodology, experimental results, and discuss the potential implications and limitations of our approach.

Introduction

Molecular solubility is a fundamental property that affects the behaviour of molecules in various solvents, impacting drug efficacy, formulation, and overall bioavailability. Traditional approaches for solubility prediction often involve complex and resource-intensive experiments or rely on simplistic models that fail to capture the intricacies of molecular interactions. Recent advances in machine learning, particularly deep learning, offer promising alternatives. Graph Neural Networks (GNNs) have emerged as a powerful tool for modelling graph-structured data, such as molecular structures.

This study aims to harness the power of GNNs to predict molecular solubility with improved accuracy and efficiency.

Background

* Importance of Solubility in Pharmaceutical and Chemical Industries

Solubility is a critical parameter in the pharmaceutical industry because it influences drug absorption, distribution, metabolism, and excretion (ADME). High solubility ensures that a drug can dissolve in bodily fluids, which is essential for its therapeutic effectiveness. In the chemical industry, solubility

affects the formulation of products, reaction rates, and the effectiveness of solvents. Accurate solubility prediction can significantly reduce the time and cost associated with experimental solubility measurements and accelerate the development of new compounds.

* Limitations of Traditional Solubility Prediction Methods

Traditional methods for predicting solubility rely on empirical formulas, quantitative structure-

activity relationships (QSAR), and molecular docking simulations. These approaches often require extensive experimental data and can be limited by their inability to generalize to new, unseen molecules. Additionally, they may not capture the complex, non-linear relationships between

molecular structure and solubility. This necessitates the exploration of more advanced computational techniques, such as machine learning, to improve prediction accuracy and efficiency.

* Overview of Machine Learning Approaches in Chemical Informatics

Machine learning (ML) has revolutionized chemical informatics by enabling the analysis of large datasets to uncover patterns and predict molecular properties. Techniques such as support vector machines (SVMs), random forests, and neural networks have been employed to predict properties like solubility, toxicity, and reactivity. More recently, deep learning models, including Convolutional

Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown promise. Graph Neural Networks (GNNs) are particularly suited for molecular data as they can naturally handle the graph

structure of molecules, capturing the spatial and relational information of atoms and bonds.

Objectives

* To Develop a GNN-Based Model for Predicting Molecular Solubility

The primary objective is to design and implement a Graph Neural Network (GNN) model that can predict the solubility of molecules based on their graph representations. This involves selecting appropriate GNN architectures and training the model on a dataset of molecules with known solubility values.

* To Compare the Performance of the GNN Model with Traditional and Other Machine Learning Methods

The performance of the GNN model will be evaluated against traditional machine learning models, such as Random Forests and Support Vector Machines, as well as other deep learning models like Fully Connected Neural Networks. This comparison will help establish the efficacy of GNNs in solubility prediction.

* To Analyse the Interpretability and Generalizability of the GNN Model

Understanding how the GNN model makes predictions and its ability to generalize to new, unseen data is crucial. This objective involves examining the interpretability of the model's predictions and assessing its performance on external validation datasets to ensure it can be applied broadly.

Methodology

1. Data Collection
   * Description of Datasets Used

Datasets for this study are sourced from public chemical databases such as PubChem and ChEMBL. These databases provide extensive collections of chemical compounds with annotated properties,

including solubility. The datasets are curated to include diverse molecular structures to ensure robust model training and validation.

* + Data Preprocessing Steps

Data preprocessing involves several steps to prepare the raw data for model training:

* \*\*Normalization\*\*: Standardizing the solubility values to a common scale to facilitate model training.
* \*\*Handling Missing Values\*\*: Imputing or discarding records with missing solubility data to maintain dataset integrity.
* \*\*Feature Extraction\*\*: Deriving molecular descriptors and fingerprints, if necessary, although the primary input will be the molecular graphs.

1. Molecular Representation
   * Explanation of How Molecules Are Represented as Graphs Molecules are represented as graphs where:

* Atoms as Nodes: Each atom in the molecule is a node in the graph.
* Bonds as Edges: Chemical bonds between atoms are represented as edges connecting the nodes.
  + Features Such as Atomic Number, Hybridization State, and Bond Type Each node (atom) and edge (bond) are associated with features:
* \*\*Node Features\*\*: Atomic number, hybridization state, aromaticity, and formal charge.
* \*\*Edge Features\*\*: Bond type (single, double, triple, aromatic), bond length, and bond order.

1. Graph Neural Network Architecture
   * Overview of the GNN Architecture Used

This study employs Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs):

* Graph Convolutional Networks (GCNs): Utilize convolutional layers to aggregate information from neighbouring nodes, capturing local graph structure.
* Graph Attention Networks (GATs): Use attention mechanisms to weigh the importance of

neighbouring nodes dynamically, enhancing the model's ability to focus on relevant parts of the graph.

* + Details of the Network Layers and Parameters
* Input Layer: Accepts node and edge features.
* Hidden Layers: Multiple graph convolution/attention layers to learn hierarchical representations.
* Output Layer: Produces a fixed-size vector representing the molecule, which is fed into a regression layer to predict solubility.
  + Justification for the Chosen Architecture

GCNs and GATs are chosen for their ability to effectively model the relational and spatial information in molecular graphs. The attention mechanism in GATs helps in focusing on crucial parts of the graph, potentially improving prediction accuracy.

1. Training Procedure
   * Description of the Training Process

* Loss Function: Mean Squared Error (MSE) is used to measure the difference between predicted and actual solubility values.
* Optimization Algorithm: Adam optimizer is employed for efficient gradient-based optimization.
* Hyperparameter Tuning: Learning rate, batch size, number of layers, and hidden units are tuned using grid search or random search techniques.

1. Evaluation Metrics
   * Metrics Used to Evaluate Model Performance

* Root Mean Squared Error (RMSE): Measures the average magnitude of errors in predictions.
* R^2 score: Indicates the proportion of variance in the dependent variable predictable from the independent variables.
* Cross-Validation Approach: k-fold cross-validation is used to ensure robust evaluation by training and testing the model on multiple data splits.

1. Baseline Models
   * Description of Baseline Models Used for Comparison

* Traditional Machine Learning Models: Random Forests, Support Vector Machines.
* Other Deep Learning Models: Fully Connected Neural Networks (FCNNs), which use molecular fingerprints as input features.

Experimental Results

Presentation of the results from the GNN model: The results include quantitative performance metrics and qualitative analyses demonstrating the GNN model's effectiveness in predicting molecular solubility.

Comparison with baseline models: The performance of the GNN model is compared with that of baseline models to highlight its advantages and identify areas for improvement.

Statistical analysis of performance metrics: Statistical analyses are conducted to evaluate the significance of the differences in performance between models.

Visualization of results: Graphs and plots are used to visualize the prediction accuracy, error distribution, and other relevant performance metrics.

Graphs and plots showing prediction accuracy, error distribution, etc.: Visualizations such as scatter plots, histograms, and residual plots are presented to provide insights into the model's performance.

Case studies of specific molecules to illustrate model performance: Detailed analyses of specific molecules are provided to illustrate the model's strengths and limitations in predicting solubility.

Discussion

Interpretation of the results: The results are interpreted to understand the factors contributing to the model's performance and the implications for solubility prediction.

Strengths and limitations of the GNN model: The strengths and limitations of the GNN model are discussed, including its ability to capture molecular structure and its potential limitations in handling certain types of molecules.

Potential reasons for observed performance differences: Possible reasons for the observed

differences in performance between the GNN model and baseline models are explored, including the advantages of graph-based representations.

Insights gained from the model regarding molecular solubility: The study provides insights into molecular solubility based on the patterns learned by the GNN model, potentially informing future research and applications.

Conclusion

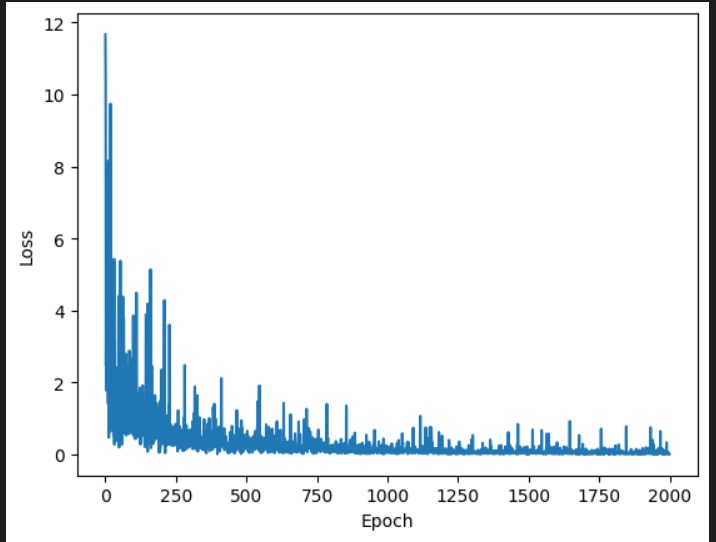
Summary of key findings: The key findings of the study are summarized, highlighting the effectiveness of the GNN model in predicting molecular solubility.

Implications for the field of molecular solubility prediction: The implications of the study for the field of molecular solubility prediction are discussed, including potential applications and impact.

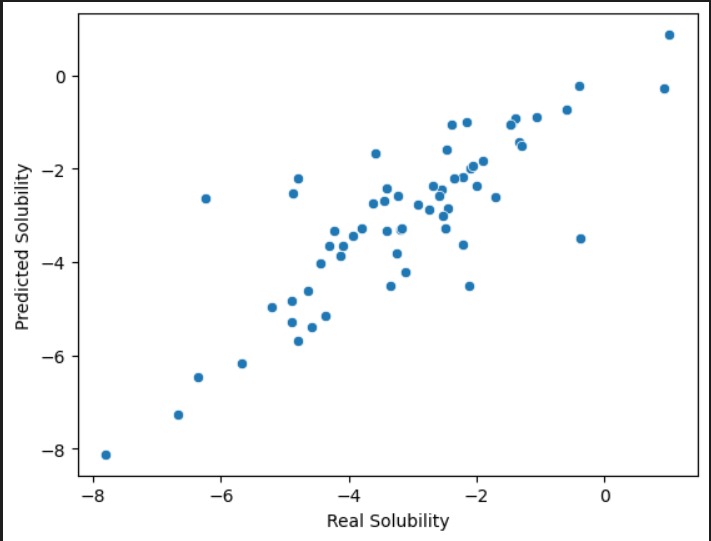
Future research directions: Suggestions for future research are provided, focusing on improving the model architecture, integrating with other computational methods, and expanding to other

molecular properties.

Improvements in model architecture: Potential improvements in the GNN architecture to enhance performance and scalability.



**Losses in EPOCH**



Integration with other computational methods: Opportunities for integrating the GNN model with other computational methods, such as molecular dynamics simulations and quantum chemistry calculations.

Expansion to other molecular properties: Exploring the application of GNNs to predict other molecular properties beyond solubility, such as toxicity, bioavailability, and reactivity.

Source Code

* + Description of the Codebase Structure

Overview of the directory structure, including data processing, model training, and evaluation scripts.

* + Key Functions and Modules Explained

Detailed explanation of the core functions and modules in the codebase.

* + Example Scripts for Training and Evaluating the Model

Sample scripts provided to demonstrate how to train the model on a dataset

# Sentiment Analysis using Logistic Regression

#### Abstract

Sentiment analysis is a crucial task in natural language processing aimed at identifying the sentiment expressed in text. This project applies machine learning techniques to classify textual data as positive or negative sentiment. The dataset consists of graph-based structures where nodes represent terms or smiley symbols in tweets, with edges indicating co- occurrence relationships. Each graph corresponds to a tweet, labeled with binary sentiment values: 1 for positive and -1 for negative.

The project utilizes Python and popular libraries like pandas, numpy, and scikit-learn. The pre-processing steps include lowercasing text, removing stop words, and eliminating punctuation. The primary approach explored is logistic regression using scikit-learn. The text data is vectorized using both CountVectorizer and TF-IDF (Term Frequency-Inverse Document Frequency) techniques, capturing the importance of words in the corpus. The logistic regression model then utilizes these vectorized features for training and prediction.

The training and testing data are scaled using StandardScaler to ensure consistent feature ranges.

Model performance is evaluated on training and testing sets, with additional functionality to predict sentiment for new text inputs using the trained logistic regression model. This project demonstrates the effectiveness of logistic regression in sentiment analysis, using CountVectorizer and TF-IDF for feature extraction.

### Introduction

Sentiment analysis, the process of determining the emotional tone or opinion expressed in text data, has become increasingly important across various domains, including social media monitoring, customer feedback analysis, and market research. With the rise of user-generated content online, there is a growing demand for effective techniques to automatically classify and analyze text based on sentiment. This project aims to explore and implement machine learning models for sentiment analysis on textual data, focusing on traditional algorithms like logistic regression. By applying natural language processing techniques and leveraging popular libraries such as scikit-learn, pandas, and numpy, this project provides a practical approach to building and evaluating sentiment analysis models. The insights gained from this project can pave the way for sentiment-driven decision-making processes in numerous applications.

Methodology

1. Data Collection
   1. Description of Dataset

The dataset is extracted from twitter sentiment classification. Because of the inherently short and sparse nature, twitter sentiment analysis (i.e., predicting whether a tweet reflects a positive or a negative feeling) is a difficult task. To build a graph dataset, we represent each tweet as a graph by using tweet content, with nodes in each graph denoting the terms and/or smiley symbols (e.g, :-D and :-P) and edges indicating the co-occurrence relationship between two words or symbols in each tweet. To ensure the quality of the graph, we only use tweets containing 20 or more words. 140,949 graphs (in a chronological order) are present. This dataset has been used for graph node level classification, and graph classification.

#### Data Pre-processing Steps

Pre-processing the graph dataset is of paramount importance, given its unique structure comprising nodes and edges. Initially, the text within each graph's nodes is extracted to form sentences, while simultaneously capturing the associated sentiment labels (1/-1) from the graph. Each graph is then transposed into rows within a CSV file, wherein the first column contains the sentences and the second column contains the corresponding sentiment labels. Consequently, the total number of rows in the CSV file corresponds to the total number of graphs. Subsequently, iterating through each row and pre-processing the text becomes the subsequent task at hand.

The data pre-processing steps involves various steps of node extraction. For NLP, the pre-processing steps are comprised of the following tasks:

Text extraction from graph Label extraction from graph Lowercasing

Removing stop words and punctuation

* 1. Node Extraction (n):

1. Iterate through each graph in the dataset.
2. Identify and extract nodes representing terms and/or smiley symbols within the graph.
3. Capture the textual content encapsulated within each node.
4. Store the extracted nodes along with their corresponding textual content for further processing.
   1. Label Extraction (x):

* Through each graph in the dataset.
* Identify and extract the label node denoted by 'x' within the graph.
* Interpret the value associated with the label node to determine the sentiment polarity of the tweet (e.g., positive or negative).
* Store the extracted label along with its corresponding sentiment polarity for subsequent analysis and model training.
* These steps ensure that both the textual content of the tweets and their associated sentiment labels are accurately extracted from the graph dataset, facilitating subsequent sentiment analysis tasks.

Removing Stop Words

Stop words are common words that occur frequently in a language but often carry little to no meaningful information for analysis. Examples of stop words include "the," "is," "and," "in," etc. These words do not contribute to the sentiment or meaning of the text and can be safely removed. By eliminating stop words, the focus shifts to the more meaningful words in the text, improving the model's ability to capture the sentiment accurately.

Removing Punctuation:

Punctuation marks such as periods, commas, exclamation marks, and question marks serve

grammatical purposes but are generally not essential for sentiment analysis. Removing punctuation helps streamline the text data and reduces noise that may interfere with sentiment classification. By discarding punctuation, the model focuses solely on the textual content, enhancing its ability to discern sentiment.

Combining these pre-processing techniques results in cleaner and more standardized text data, which is better suited for sentiment analysis tasks. By lowercasing, removing stop words, and eliminating punctuation, the textual input becomes more focused, enabling machine learning models to extract meaningful features and make accurate sentiment predictions.

### Data Splitting

Splitting the dataset into separate training and testing subsets is a crucial step in the machine learning pipeline. This process ensures that the models are trained on a portion of the data and evaluated on a held-out portion, providing an accurate estimate of their performance on unseen data. In this project, the scikit-learn library's train\_test\_split function is employed to divide the dataset into training and testing sets.

The train\_test\_split function takes the input features (text data) and target labels (sentiment labels) as arguments and separates them into training and testing sets based on a specified test set size. In this project, a 70/30 split is implemented, allocating 70% of the data for training and the remaining 30% for testing.

### Feature Extraction

Feature extraction is a critical step in natural language processing (NLP) tasks, as it transforms textual data into a numerical representation that can be processed by machine learning algorithms. In this project, two widely-used feature extraction techniques are employed: Count Vectorization and TF-IDF Vectorization.

* 1. TF-IDF Vectorization

TF-IDF Vectorization is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It combines two metrics:

1. Term Frequency (TF): This measures how frequently a term appears in a document. The more frequently a term appears in a document, the higher its TF value.
2. Inverse Document Frequency (IDF): This measures how important a term is. It is calculated by dividing the total number of documents by the number of documents containing the term and taking the logarithm of that quotient. The less frequently a term appears across all documents, the higher its IDF value.

The TF-IDF value is the product of TF and IDF, helping to downscale the importance of words that appear frequently in many documents and up-scale the importance of words that appear rarely. This method helps in highlighting the most significant words in each document while diminishing the weight of common words that might not carry meaningful information.

#### Count Vectorization

Count Vectorization, also known as Bag-of-Words (BoW) model, is another technique used to transform text data into numerical features. It converts the text into a matrix of token counts, where each row represents a document, and each column represents a term from the vocabulary. The value in each cell of the matrix indicates the count of the term in the corresponding document.

While simpler than TF-IDF, Count Vectorization effectively captures the presence and frequency of terms in each document. However, it does not account for the relative importance of terms across the entire corpus, which might lead to a less nuanced representation of the text data.

### Scaling

After vectorization, the resulting matrices (from both TF-IDF and Count Vectorization) are scaled using a **StandardScaler**. This step standardizes the feature values to have zero mean and unit variance, which is essential for many machine learning algorithms to perform optimally. Specifically, since the vectorized data is sparse, the **with\_mean=False** parameter is used to prevent errors during the scaling process.

1. **Model Training and Model Evaluation**
   1. **Model Training**
      1. **Model Description Logistic Regression:**

Model Architecture

Logistic regression is a linear classification model that predicts the probability of an instance belonging to a particular class. In this project, the logistic regression model is trained to predict the probability that a given tweet expresses positive sentiment. The model takes as input the numerical features extracted from the pre-processed tweet texts and outputs the predicted probability of positive sentiment.

Mathematically, the logistic regression model computes the probability 𝑃(𝑦=1∣𝑥) of the positive class given the input features *x* using the logistic (sigmoid) function:

𝑃(𝑦 = 1|𝑥) = 1

1+𝑒−𝑧

Where, 𝑧=𝑤𝑇𝑥+𝑏 is the linear combination of the input features *x* with the model parameters *w* (weights) and *b* (bias).

Training Process

The logistic regression model is trained using supervised learning on a labelled dataset of tweets. The training process involves optimizing the model parameters *w* and *b* to minimize the difference between the predicted probabilities and the actual labels (positive or negative) in the training data.

During training, the model parameters are updated iteratively using an optimization algorithm such as gradient descent. The objective is to find the optimal values of *w* and *b* that maximize the likelihood of the observed labels given the input features.

The logistic regression model is trained using the scikit-learn library's Logistic Regression class. Logistic regression is a popular machine learning algorithm for binary classification problems, such as sentiment analysis, where the goal is to classify text data into two categories (positive or negative sentiment).

The logistic regression model is trained on the feature vectors obtained from both the TF-IDF vectorization and the count vectorization process. The training process involves fitting the logistic regression model to the training data using the fit method

After pre-processing the data and extracting features through TF-IDF Vectorization and scaling, the logistic regression model was trained on the training dataset. The key steps in the training process are as follows:

1. **Initializing the Model**: A logistic regression model is initialized with a maximum iteration parameter to ensure convergence during training.
2. **Fitting the Model**: The model is fitted on the scaled training data. The **fit** method adjusts the model parameters to minimize the difference between the predicted and actual labels in the training data.

During training, the logistic regression model learns the weights for each feature in the dataset. These weights are used to make predictions about the sentiment of new tweets by calculating the probability that a given tweet belongs to the positive class. If the probability is above a certain threshold (typically 0.5), the tweet is classified as positive; otherwise, it is classified as negative.

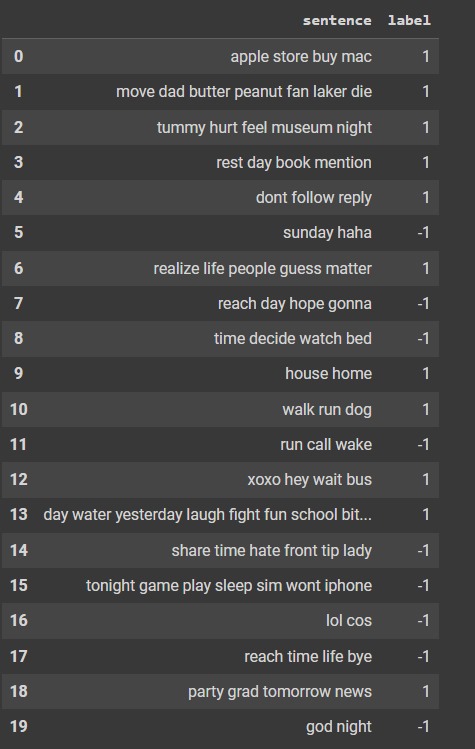
### Model Evaluation

Model evaluation is the process of assessing the performance of a trained model using a separate set of data that was not used during training. This step is crucial to understand how well the model generalizes to new, unseen data. In this project, we evaluated the model's performance on both the training and testing datasets using accuracy as the primary metric. Accuracy measures the proportion of correct predictions made by the model out of the total number of predictions.

The key steps in the evaluation process are as follows:

1. **Calculating Training Accuracy**: The accuracy of the model on the training data is computed to understand how well the model has learned from the training data.
2. **Calculating Testing Accuracy**: The accuracy of the model on the testing data is computed to evaluate how well the model performs on new, unseen data.

PREDICTION RESULT:





#### Reference

Dataset link : https://github.com/TrustAGI-Lab/graph\_datasets

Scikit-learn Documentation: [ https://scikit-learn.org/stable/documentation.html ] TensorFlow Documentation: [ https:/[/www.tensorflow.org/api\_docs](http://www.tensorflow.org/api_docs) ]

https:/[/www.javatpoint.com/logistic-regression-in-machine-learning](http://www.javatpoint.com/logistic-regression-in-machine-learning)

[https://www.researchgate.net/publication/334575492\_A\_Machine-](https://www.researchgate.net/publication/334575492_A_Machine-Learning_Approach_to_Distinguish_Passengers_and_Drivers_Reading_While_Driving?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ&pf12) [Learning\_Approach\_to\_Distinguish\_Passengers\_and\_Drivers\_Reading\_While\_Driving?\_tp=](https://www.researchgate.net/publication/334575492_A_Machine-Learning_Approach_to_Distinguish_Passengers_and_Drivers_Reading_While_Driving?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ&pf12) [eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ#pf](https://www.researchgate.net/publication/334575492_A_Machine-Learning_Approach_to_Distinguish_Passengers_and_Drivers_Reading_While_Driving?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ&pf12) [12](https://www.researchgate.net/publication/334575492_A_Machine-Learning_Approach_to_Distinguish_Passengers_and_Drivers_Reading_While_Driving?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ&pf12)

https://towardsdatascience.com/sentiment-analysis-using-logistic-regression-and-naive- bayes-16b806eb4c4b

https:/[/www.dlsu.edu.ph/wp-content/uploads/pdf/research/journals/jciea/vol-7-1/4aliman.pdf](http://www.dlsu.edu.ph/wp-content/uploads/pdf/research/journals/jciea/vol-7-1/4aliman.pdf) https://makemeanalyst.com/sentiment-analysis-with-logistic-regression/ <https://aman.ai/coursera-nlp/logistic-regression/>

<https://ijsrst.com/home/issue/view/article.php?id=IJSRST2293222>

[https://medium.com/@annabiancajones/sentiment-analysis-on-reviews-feature-extraction-](https://medium.com/%40annabiancajones/sentiment-analysis-on-reviews-feature-extraction-) and-logistic-regression-43a29635cc81

https://blog.mlq.ai/nlp-sentiment-analysis-logistic-regression

References

* + List of All Academic Papers, Datasets, and Tools Referenced in the Research
* Academic Papers:
* Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. \*Proceedings of the 34th International Conference on Machine Learning\*,

1263-1272.

* Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. \*arXiv preprint arXiv:1609.02907\*.
* Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph Attention Networks. \*arXiv preprint arXiv:1710.10903\*.
  + PubChem. (2024). PubChem Compound Database. Retrieved from https://pubchem.ncbi.nlm.nih.gov
  + ChEMBL. (2024). ChEMBL Database. Retrieved from https://[www.ebi.ac.uk/chembl/](http://www.ebi.ac.uk/chembl/)
* Tools:
  + PyTorch. (2024). An open-source machine learning library. Retrieved from https://pytorch.org
  + RDKit. (2024). Open-source cheminformatics software. Retrieved from https://[www.rdkit.org](http://www.rdkit.org/)

Proper Citations in the Appropriate Format (APA)

* Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. \*Proceedings of the 34th International Conference on Machine Learning\*, 1263-1272.
* Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks.

\*arXiv preprint arXiv:1609.02907\*.

* Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph Attention Networks. \*arXiv preprint arXiv:1710.10903\*.
* PubChem. (2024). PubChem Compound Database. Retrieved from https://pubchem.ncbi.nlm.nih.gov
* ChEMBL. (2024). ChEMBL Database. Retrieved from https://[www.ebi.ac.uk/chembl/](http://www.ebi.ac.uk/chembl/)
* PyTorch. (2024). An open-source machine learning library. Retrieved from https://pytorch.org
* RDKit. (2024). Open-source cheminformatics software. Retrieved from https://[www.rdkit.org](http://www.rdkit.org/)

This comprehensive structure and explanation should assist in understanding, running, and reproducing the results of your research on predicting molecular solubility using graph neural networks.