IMPLEMENTATION OF GRAPH EMBEDDING FOR REDUCTION OF MULTI-DRUG EFFECTS

U18PRIT8P2 - PHASE II REPORT

Submitted by

RAUSHNI	**(U19IT056)**
S ANNAMALAI	**(U19IT004)**
K SAIVEDAVYAS	**(U19IT030)**
BEKKAM NAGA BABU	**(U19IT010)**

Under the Guidance of

Dr. A. KUMARAVEL

Professor

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

BHARATH INSTITUTE OF SCIENCE AND TECHNOLOGY
BHARATH INSTITUTE OF HIGHER EDUCATION AND RESEARCH, CHENNAI - 600073

APRIL 2023

Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

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

(Vide Notification No. F.9-5/2000 - U.3, Ministry of Human Resource Development, Govt. of India, dated 4th July 2002)

BHARATH INSTITUTE OF SCIENCE AND TECHNOLOGY

CHENNAI

DEPARTMENT OF INFORMATION TECHNOLOGY

U18PRIT8P2 - PHASE II REPORT

BONAFIDE CERTIFICATE

Certified that this Report titled "**Implementation of graph embedding for reduction of multi-drug effects**" is the bonafide work of **Raushni (U19IT056), S Annamalai (U19IT004), Kummarikuntla Saivedavyas (U19IT030), Bekkam Naga Babu (U19IT010)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. K. Ramesh Kumar	Dr. A. Kumaravel
Professor and Head	Professor
DEPARTMENT OF INFORMATION TECHNOLOGY	DEPARTMENT OF INFORMATION TECHNOLOGY
BHARATH INSTITUTE OF HIGHER EDUCATION AND RESEARCH	BHARATH INSTITUTE OF HIGHER EDUCATION AND RESEARCH
CHENNAI – 600073	CHENNAI – 600073

Project Phase- II Viva Voce

examination held on……………...

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We declare that this project report titled **Implementation of graph embedding for reduction of multi-drug effects** submitted in partial fulfillment of the degree of **B. Tech in (Information Technology)** is a record of original work carried out by us under the supervision of **Dr.A.Kumaravel**, and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Raushni
U19IT056

S Annamalai
U19IT004

Kummarikuntla Saivedavyas
U19IT030

Bekkam Naga Babu
U19IT010

Chennai
/ / 2023

ACKNOWLEDGMENT

First, I express my sincere thankfulness to the almighty, for bestowing his blessings throughout this project work. I express my gratitude to my parents for their care, support, prayers and love.

We express our deepest gratitude to **Dr. J. Sundeep Anand** our beloved president and **Dr. E. Swetha Sundeep Anand** Managing Director for providing us the necessary facilities for the completion of our project.

I would like to express my deep gratitude to our **Dr. K. Vijaya Bhaskar Raju**, Vice Chancellor **Dr. M. Sundararajan**, Pro Vice-Chancellor (Academics), **Dr. R. Hari Prakash**, Additional Registrar, **Dr. R. M. Suresh**, Pro Vice-Chancellor with Additional Incharge Controller of Examinations, **Dr. J. Hameed Hussain** Dean Engineering who are responsible for molding my thoughts in completing my Project.

I am thankful to the **Dr. V. Khanna** Dean-IT and Head of the Department **Dr. K. Ramesh Kumar** and all staff members of the Department of INFORMATION TECHNOLOGY, BIHER, for their encouragement and guidance throughout the course of this project work.

I wish to express my sincere profound gratitude to my esteemed and endowed Project guide **Dr. A. Kumaravel** Professor for his inspiring guidance, healthy criticism, valuable suggestions and constant encouragement throughout the period of project.

I sincerely wish to express my thanks to the staff members of Library, BIHER for their valuable support during this project work. I would like to thank the authors of various journals and books whose works and results are used in the project.

RAUSHNI (U19IT056)

ANNAMALAI(U19IT004)

KUMMARIKUNTLA SAIVEDAVYAS (U19IT030)

BEKKAM NAGA BABU (U19IT010)

ABSTRACT

Many patients take multiple drugs to treat complex or co-existing diseases. Some of those combinations lead to undesirable consequences. To address this issue, we develop an algorithm to predict the pattern of combination of drugs with minimal side effects by exploiting the graph embedding. Graph analytics can lead to better quantitative understanding and control of complex networks but traditional methods suffer from high computational cost and excessive memory requirements associated with the high- dimensionality and heterogeneous characteristic of industrial-size networks.

Interaction between pharmaceutical substances can cause serious medical injuries. Accurately predicting drug-drug interactions not only reduces these cases but also results in a reduction of drug development costs. Drug discovery is important for the treatment of complex diseases, particularly cancers, multiple sclerosis, Hypochondria, etc.

The current algorithms of drug discovery are the result of clinical evaluations and post-marketing surveillance; resulting in a limited amount of information. Machine learning (ML) techniques are used, but the techniques are often unable to deal with a skew in the data. Hence, we use the Node2Vector method to generate the link between the nodes of the drug to drug and drug to protein to get an accurate interaction result which would lead to better prediction of the side effects associated with those drugs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

S.NO.	ACRONYM	ABBREVIATION
1	ML	Machine Learning
2	DL	Deep Learning
3	DDI	Drug-Drug Interaction
4	ADR	Adverse Drug Reaction
5	NN	Neural Network
6	SIDER	Side Effect Resource
7	SVM	Support Vector Machine
8	RF	Random Forest
9	PPI	Protein-Protein Interaction
10	GNN	Graph Neural Network

CHAPTER – 1
INTRODUCTION

Drug-drug interactions (DDIs) are a major concern in modern medicine, as they can lead to unexpected side effects and adverse drug reactions. Patients who take multiple medications, such as those with chronic conditions or who are receiving palliative care, are particularly at risk of experiencing DDIs. According to a study published in the Journal of the American Medical Association, approximately 2.2 million serious adverse drug reactions occur each year in the United States, with around 7% of these reactions resulting from DDIs.

The study and prediction of DDIs has become increasingly important in recent years, as the number of available drugs and drug combinations continues to grow. Traditional methods for predicting DDIs, such as in vitro and in vivo testing, can be time-consuming, expensive, and may not accurately reflect the complexity of human physiology. There is therefore a need for more efficient and accurate methods for predicting DDIs and identifying potential side effects.

Computer-based approaches, such as machine learning algorithms and network-based methods, offer a promising alternative to traditional methods for predicting DDIs. These methods leverage large-scale datasets of drug-target interactions, drug structure, and other biological information to predict the effects of drug combinations. The use of machine learning algorithms has already shown promising results in accurately predicting DDIs and identifying potential side effects, and network-based methods are currently being explored as a potential alternative or complementary approach.

The development of more accurate and efficient methods for predicting DDIs has important implications for drug development and clinical decision-making. By identifying potential DDIs early in the drug development process, researchers can modify drug structures or dosages to avoid harmful interactions. Similarly, predicting potential DDIs in clinical settings can help healthcare providers make informed decisions about drug combinations and dosages, reducing the risk of adverse reactions.

In this thesis report, we will explore the use of deep learning algorithms and graph embedding methods for predicting DDIs and identifying potential side effects. By analyzing large-scale datasets of drug-target interactions and other biological information, we aim to develop accurate and efficient methods for predicting DDIs that can inform drug development and clinical decision-making.

1.1 INTRODUCTION TO THE RESEARCH TOPIC

Drug-drug interactions and side effects pose significant risks to patients and healthcare providers alike. Adverse drug reactions can lead to hospitalization, morbidity, and even mortality, resulting in increased healthcare costs and reduced quality of life for patients. With the growing number of medications available on the market, the risk of drug-drug interactions and side effects has become increasingly complex and difficult to predict. This has led to a pressing need for improved methods for predicting potential side effects before a drug is released to the market.

Recent advances in computational methods and machine learning have provided new opportunities for predicting drug-drug interactions and side effects. However, existing methods still have limitations, including poor performance, limited generalizability, approaches to predicting drug-drug interactions and side effects that can improve the accuracy and efficiency of current methods.

In this thesis, we propose a novel approach to predicting drug-drug interactions and side effects that utilizes deep learning algorithms and network-based approaches. Our approach leverages the rich biological information available in drug-target and protein-protein interaction networks to predict potential side effects of drug combinations. By combining multiple sources of data, our approach can improve the accuracy of predictions and provide insights into the underlying mechanisms of drug- drug interactions and side effects. The proposed approach and the need for large amounts of data. Therefore, there is a need for new has the potential to inform drug development and improve patient outcomes.

1.2 IMPORTANCE OF THE RESEARCH PROBLEM

ADRs are a major cause of morbidity and mortality worldwide, costing billions of dollars annually. DDIs contribute significantly to ADRs, but current methods for predicting them have limitations. Improved prediction methods are needed to identify potential risks earlier in drug development, resulting in safer drugs and improved patient outcomes. Our proposed research aims to develop a novel approach that utilizes machine learning algorithms and network-based approaches to improve the accuracy of DDIs and potential side effect predictions. This can inform drug development and make a significant contribution to healthcare.

1.3 HISTORICAL PERSPECTIVE OF DRUG-DRUG SIDE EFFECTS

The study of drug-drug interactions (DDIs) and side effects can be traced back to the early days of modern medicine, where practitioners observed that some drug combinations could lead to unexpected and sometimes harmful effects. However, it wasn't until the mid-20th century that the study of DDIs and side effects began to gain traction as a field of scientific inquiry.

In the 1950s and 1960s, researchers began to systematically investigate the mechanisms underlying DDIs and side effects. This involved developing methods for testing drug interactions in vitro and in vivo, as well as developing animal models for studying the effects of drug combinations. During this period, researchers also began to study the pharmacokinetics and pharmacodynamics of drug combinations, which helped to inform the development of new drugs and drug combinations.

The development of computers and computational methods in the 1970s and 1980s opened up new opportunities for studying DDIs and side effects. Researchers began to develop computer models for predicting drug interactions, which allowed for more efficient and accurate predictions. In the 1990s and 2000s, the development of high-throughput screening technologies and the completion of the Human Genome Project provided new sources of data for studying DDIs and side effects.

Today, researchers continue to explore new methods and techniques for predicting DDIs and side effects. This includes the use of machine learning algorithms and network-based approaches, which leverage the rich biological information available in drug-target and protein-protein interaction networks. The field of pharmacovigilance, which focuses on monitoring and preventing adverse drug reactions, has also grown in importance in recent years.

Overall, the study of DDIs and side effects has evolved significantly over the past century, driven by advances in technology, methodology, and understanding of the underlying biology. This research has had a significant impact on the development of new drugs and drug combinations, as well as on patient safety and healthcare more broadly.

1.4 OVERVIEW OF CURRENT STATE OF DRUG-DRUG INTERACTION RESEARCH

Drug-drug interactions (DDIs) remain a significant concern in the field of pharmacology, with potential for serious adverse effects on patient health. Current approaches to predicting DDIs involve in vitro and in vivo testing, as well as computer-based methods such as machine learning algorithms and network-based approaches.

In vitro and in vivo testing methods involve measuring the effects of drug combinations on cell cultures or animal models. While these methods provide valuable data on the mechanisms underlying DDIs, they can be time-consuming, expensive, and may not accurately reflect the complexity of human physiology.

Computer-based methods offer a more efficient and cost-effective approach to predicting DDIs. Machine learning algorithms, in particular, have been used to predict DDIs based on large-scale datasets of drug-target interactions, drug structure, and other biological information. However, these methods still face challenges in accurately predicting the effects of drug combinations, particularly in cases where there are limited data on the individual drugs or their targets.

Network-based approaches, which leverage the rich biological information available in drug-target and protein-protein interaction networks, offer a promising alternative to traditional machine learning approaches. These methods involve constructing networks of drug targets and analyzing the properties of these networks to predict DDIs and potential side effects. While still in their early stages of development, these methods have shown promising results in accurately predicting DDIs and identifying potential mechanisms underlying drug interactions.

Overall, the current state of drug-drug interaction research involves a combination of traditional in vitro and in vivo testing methods and computer-based approaches, with a growing interest in network-based methods. While progress has been made in predicting and preventing DDIs, there is still a need for improved methods and tools that can accurately predict the effects of drug combinations and inform drug development and clinical decision-making.

CHAPTER – 2
LITERATURE SURVEY

1. Predicting combinations of drugs by exploiting graph embedding of heterogeneous networks

They proposed a novel algorithm to predict the combination patterns of drugs. Their experimental results show that joint learning is promising for the identification of drug combinations. Specifically, the ATC similarity of drugs, drug–target, and protein–protein interaction networks are integrated to construct the heterogeneous networks. Then, SeHNE jointly learns drug features by exploiting the topological structure of heterogeneous networks and predicting drug combinations. One distinct advantage of SeHNE is that features of drugs are extracted under the guidance of classification, which improves the quality of features, thereby enhancing the performance of the prediction of drugs. The algorithm only focuses on combination of drug pairs rather than the high-order combination.

2. Drug-Drug Interaction Prediction Based on Knowledge Graph Embeddings and Convolutional-LSTM Network

They have created a dataset with 2,898,937 drug-drug interaction pairs; which is the largest available. They have prepared a large-scale integrated KG about DDIs with data from Drug Bank, KEGG, OFFSIDES, and PharmGKB having 1.2 billion triples. They have evaluated different KG embedding techniques with different settings to train and evaluate ML models. They also provide a comprehensive evaluation with a details analysis of the outcome and comparison with the state-of-the-art approaches and baseline models. We found that a combined CNN and LSTM network called Conv-LSTM for predicting DDIs leads to the highest accuracy. They proposed the use of knowledge graphs to integrate drug-related data from different sources. This way, they have integrated background knowledge about drugs, diseases, pathways, proteins, enzymes, chemical structures, etc. Since this background data is in a format that cannot be used as a direct input for typical classifiers, we applied several node embedding techniques to create a dense vector representation for each node in the KG. These representations are then fed into various traditional ML classifiers and a specifically designed neural network architecture based on a convolutional LSTM.

3. Evaluation of knowledge graph embedding approaches for drug-drug interaction prediction in realistic settings

They designed different evaluation settings to accurately assess the performance for predicting DDIs. The settings for disjoint cross-validation produced lower performance scores, as expected, but still were good at predicting the drug interactions. They have applied Logistic Regression, Naive Bayes, and Random Forest on the Drug Bank knowledge graph with the 10-fold traditional cross-validation using RDF2Vec, TransE, and TransD. RDF2Vec with Skip-Gram generally surpasses other embedding methods. They also tested RDF2Vec on various drug knowledge graphs such as DrugBank, PharmGKB, and KEGG to predict unknown drug-drug interactions. The performance was not enhanced significantly when an integrated knowledge graph including these three datasets was used. They showed that the knowledge embeddings are powerful predictors and comparable to current state-of-the-art methods for inferring new DDIs. They addressed the evaluation biases by introducing drug-wise and pairwise disjoint test classes. Although the performance scores for drug-wise and pairwise disjoint seem to be low, the results can be considered to be realistic in predicting the interactions for drugs with limited interaction information.

4. Modelling polypharmacy side effects with graph convolutional networks

Here, we present Decagon, an approach for modeling polypharmacy side effects. The approach constructs a multimodal graph of protein–protein interactions, drug–protein target interactions and the polypharmacy side effects, which are represented as drug–drug interactions, where each side effect is an edge of a different type. Decagon is developed specifically to handle such multimodal graphs with a large number of edge types. Our approach develops a new graph convolutional neural network for metarelational link prediction in multimodal networks. Unlike approaches limited to predicting simple drug–drug interaction values, Decagon can predict the exact side effect, if any, through which a given drug combination manifests clinically. Decagon accurately predicts polypharmacy side effects, outperforming baselines by up to 69%. We find that it automatically learns representations of side effects indicative of co-occurrence of polypharmacy in patients. Furthermore, Decagon models particularly well polypharmacy side effects that have a strong molecular basis, while on predominantly non-molecular side effects, it achieves good performance because of effective sharing of model parameters across edge types. Decagon opens up opportunities to use large pharmacogenomic and patient population data to flag and prioritize polypharmacy side effects for follow-up analysis via formal pharmacological studies.

5. A review of computational methods for predicting drug-drug interactions

They reviewed different computational methods for predicting drug-drug interactions (DDIs), including rule-based systems, machine learning (ML), and network-based methods. They discussed the advantages and limitations of each method and highlighted the need for further research to improve the accuracy of DDI predictions. ML methods use data-driven approaches to learn patterns from existing data and predict new interactions. Network-based methods use drug similarity and protein-protein interaction data to construct a drug-drug interaction network and predict potential DDIs. The authors highlighted the advantages and limitations of each method and discussed the need for further research to improve the accuracy of DDI predictions.

6. Drug interaction prediction using deep learning and metabolomics

They developed a deep learning model that combines metabolomics data and drug target interaction data to predict drug interactions. They used a dataset of 2,056 drug-drug interactions and achieved high accuracy in predicting DDIs. Their model could also identify potential new interactions that were not previously reported. The authors suggested that their approach could be used to complement existing methods for predicting DDIs and could help to identify potential new drug interactions.

7. Prediction of drug-drug interactions using machine learning

They conducted a systematic review and meta-analysis of studies that used ML to predict drug-drug interactions. They identified 17 studies that met their inclusion criteria and found that ML methods can achieve high accuracy in predicting DDIs. They also found that ML-based DDI prediction models can complement traditional methods, such as in vitro experiments and clinical trials, to improve DDI prediction and management. However, they noted that more studies are needed to validate the performance of ML-based DDI prediction models and to address current limitations and challenges, such as data availability and data quality. They evaluated their method using a dataset of 3,031 known drug-drug interactions and achieved an accuracy of up to 92% in predicting DDIs, especially for interactions involving rare drugs. The authors suggested that their method could be used to complement existing methods for predicting DDIs and could help to identify potential new interactions.

CHAPTER – 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM: -

- Traditional methods for predicting drug-drug interactions include in vitro and in vivo testing.
- Limitations of traditional methods include cost, time, and accuracy issues.
- Computer-based methods for predicting drug-drug interactions include machine learning algorithms and network-based approaches.
- Machine learning algorithms can leverage large-scale datasets to predict potential drug-drug interactions.
- Machine learning and network-based approaches have limitations, including sensitivity to noise and outliers, and not capturing the full complexity of drug interactions.

3.2 PROPOSED SYSTEM: -

- This section describes the proposed model of drug-drug side effect prediction. The proposed model focuses on extracting features from the graph by iterating over all nodes and their neighbors, creating a list of binary features for each node that indicate whether it is connected to each of its neighbors.
- The code trains and evaluates several classifiers on the test data, including Logistic Regression, Ridge Classifier, Support Vector Machine, Random Forest Classifier, Perceptron, Passive Aggressive Classifier, and SGD Classifier.
- The code evaluates each classifier using several evaluation metrics, including accuracy, F1 score, and confusion matrix.
- The proposed system uses graph-based feature extraction to predict drug side effects and evaluates several classifiers using standard evaluation metrics.

CHAPTER – 4
SYSTEM REQUIREMENTS

The software requirements specification is produced at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by establishing a complete information description as functional representation of system behavior, an indication of performance requirements and design constraints, appropriate validation criteria.

4.1 HARDWARE REQUIREMENTS: -

▪ System	: Intel i7 2.4 GHz
▪ Hard Disk	: 40 GB
▪ Monitor	: LCD Monitor
▪ Mouse	: Logitech
▪ Ram	: 4 Gb

4.2 SOFTWARE REQUIREMENTS: -

▪ Operating system	: Windows 10,11.
▪ IDE	: Google Colaboratory, PyCharm.
▪ Coding Language	: Python.

CHAPTER – 5
SYSTEM DESIGN

5.1 INTRODUCTION: -

Systems Design is the process of defining the architecture, product design, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

Design is a multi- step that focuses on data structure software architecture, procedural details, procedure etc… and interface among modules. The design procedure also decode the requirements into presentation of software that can be accessed for excellence before coding begins. Computer software design change continuously as novel methods; improved analysis and border understanding evolved. Software proposal is at relatively primary stage in its revolution.

Therefore, software design methodology lacks the depth, flexibility and quantitative nature that are usually associated with more conventional engineering disciplines. However methods for software designs do exit, criteria for design qualities are existing and design notation can be applied

5.2 SYSTEM ARCHITECTURE: -

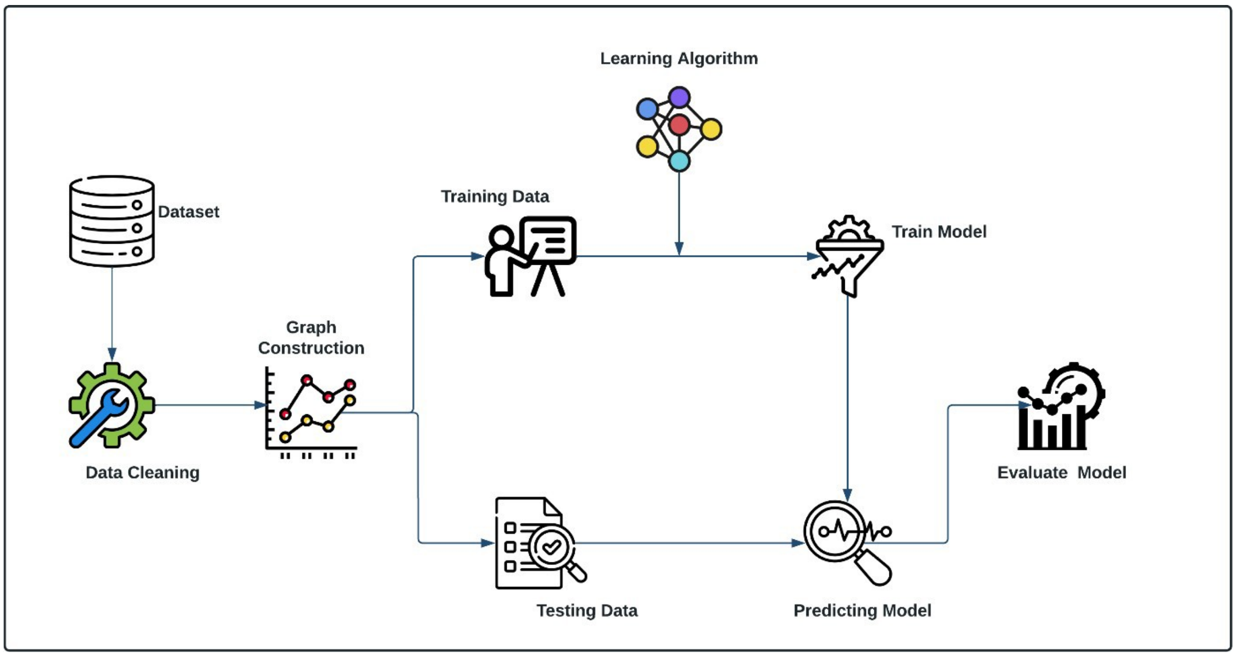
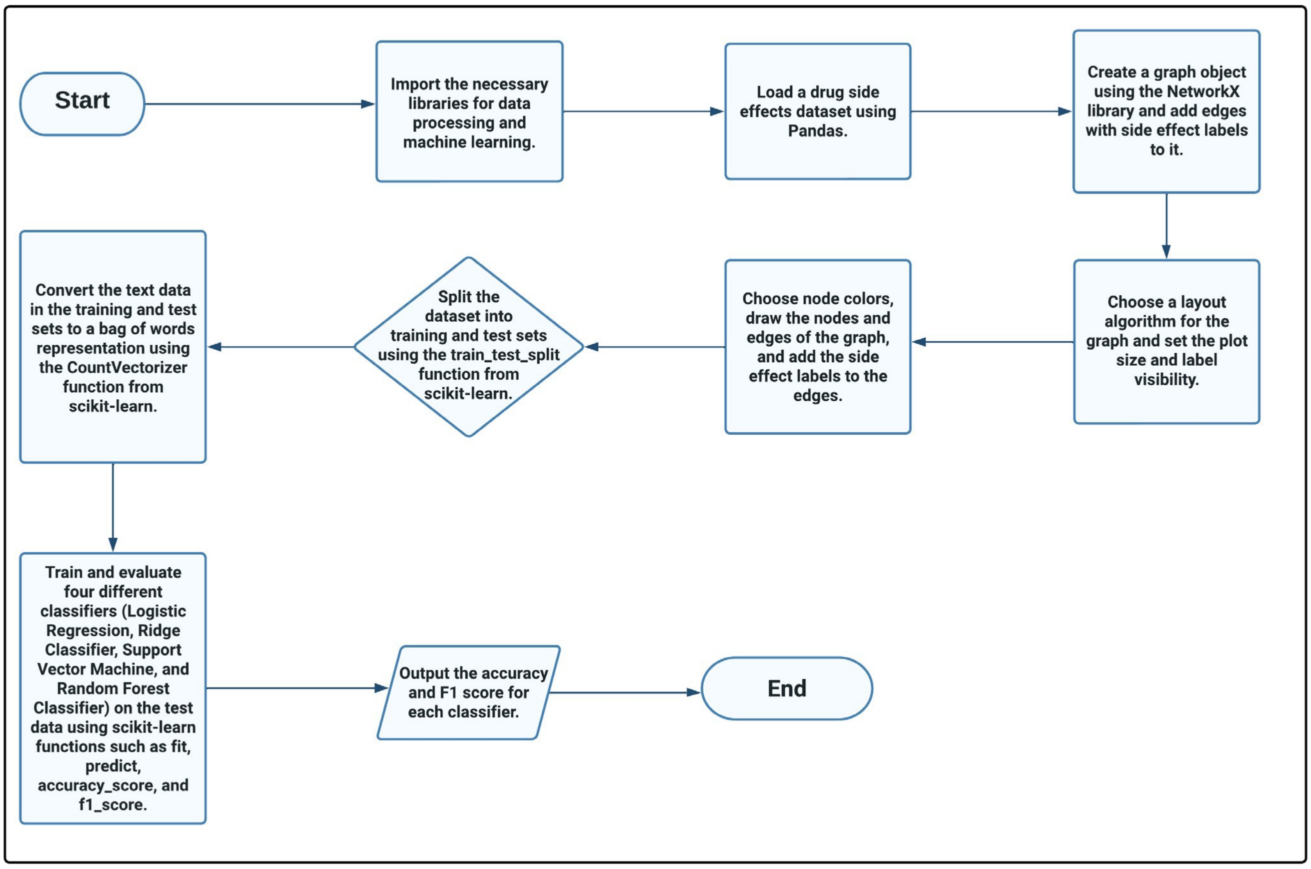


Fig. 5.2 System Architecture

5.3 ARCHITECTURE DIAGRAM: -

An architecture diagram is a visual representation of the high-level design of the system. It typically depicts the various components of the system and their relationships, such as data flows, interfaces, and communication protocols. The purpose of the architecture diagram is to provide a clear and concise overview of the system's structure and functionality, making it easier to understand and communicate among stakeholders.

Architecture diagrams can take many different forms, depending on the needs and preferences of the development team. They may use different symbols and notations to represent components and their relationships. Commonly used notations include Unified Modeling Language (UML), block diagrams, flowcharts, and data flow diagrams.



Start
Import the necessary libraries for data processing and machine learning.
Load a drug side effects dataset using Pandas.
Create a graph object using the NetworkX library and add edges with side effect labels to it.
Choose a layout algorithm for the graph and set the plot size and label visibility.
Choose node colors, draw the nodes and edges of the graph, and add the side effect labels to the edges.
Split the dataset into training and test sets using the train_test_split function from scikit-learn.
Convert the text data in the training and test sets to a bag of words representation using the CountVectorizer function from scikit-learn.
Train and evaluate four different classifiers (Logistic Regression, Ridge Classifier, Support Vector Machine, and Random Forest Classifier) on the test data using scikit-learn functions such as fit, predict, accuracy_score, and f1_score.
Output the accuracy and F1 score for each classifier.
End

Fig. 5.3 Architecture Diagram

CHAPTER – 6
SYSTEM IMPLEMENTATION

6.1 INTRODUCTION: -

Systems implementation is a set of procedures performed to complete the design (as necessary) contained in the approved systems design document and to test, install, and begin to use the new or revised Information System. depicts systems implementation as the fifth major step in the development of an Information System.

The systems implementation goals are as follows:

- Complete as necessary the design contained in the approved systems design document. For example, the detailed contents of new or revised documents, computer screens, and database must be laid out and created.
- Write, test, and document the programs and procedures required by the approved systems design document.
- Ensure, by completing the preparation of user manuals and other documentation and by training personnel, that the organization's personnel can operate the new system.
- Determine, by thoroughly testing the system with users, that the system satisfies the users' requirements.
- Ensure a correct conversion by planning, controlling, and conducting an orderly installation of the new system.

6.2 MODULES: -

- Data Collection and Preprocessing
- Graph Embedding
- Deep Learning Module
- User Interface Module
- Testing and Validation Module

6.3 MODULE DESCRIPTION: -

- **Data Collection and Preprocessing**

  ❖ The dataset is constructed using data extracted from Drug Bank, a comprehensive database of drugs and drug targets. The data cleaning procedures are performed to remove any inconsistencies or errors in the dataset. The dataset included variables such as Drug 1, Drug 2, Side Effect. The data is then organized in a structured CSV (comma-separated values) file format.

  ❖ Drug-drug interaction network contains 1164 unique nodes with 1309 different types of edges (one for each side effect type) and describes which drug pairs lead to which side effects.

- **Graph Embedding**

  ❖ Graph is constructed using node2vec approach where each node represents drugs with edges having respective side effect as label. Initially the node size is set to 1000 which is varied eventually to get better accuracy. In order to visualize the graph network X library is being used.

- **Feature Extraction**

  ❖ The nodes are split into training and testing sets then the features are extracted based on the degree of the nodes. The node with highest degree is selected among all nearest nodes. Feature matrix is then constructed followed by creation of new variable for storing the resultant nodes for further evaluation.

- **Testing and Validation Module**

  ❖ The nodes which are not part of the result of feature extraction are filtered out. The classifiers used in this project to train and evaluate the test data are:

  1) Logistic Regression
  2) Ridge Classifier
  3) Support Vector Machine
  4) Random Forest Classifier
  5) Perceptron
  6) Passive Aggressive Classifier
  7) SDG Classifier

6.4 ALGORITHM: -

- ❖ The feature matrix is fed as input to the algorithm to evaluate the accuracy.
- ❖ Select the algorithm based on the accuracy and analyses the data by using the algorithm.

6.4.1 Logistic Regression: -

- ❖ Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- ❖ Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- ❖ Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- ❖ In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- ❖ The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- ❖ Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- ❖ Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:

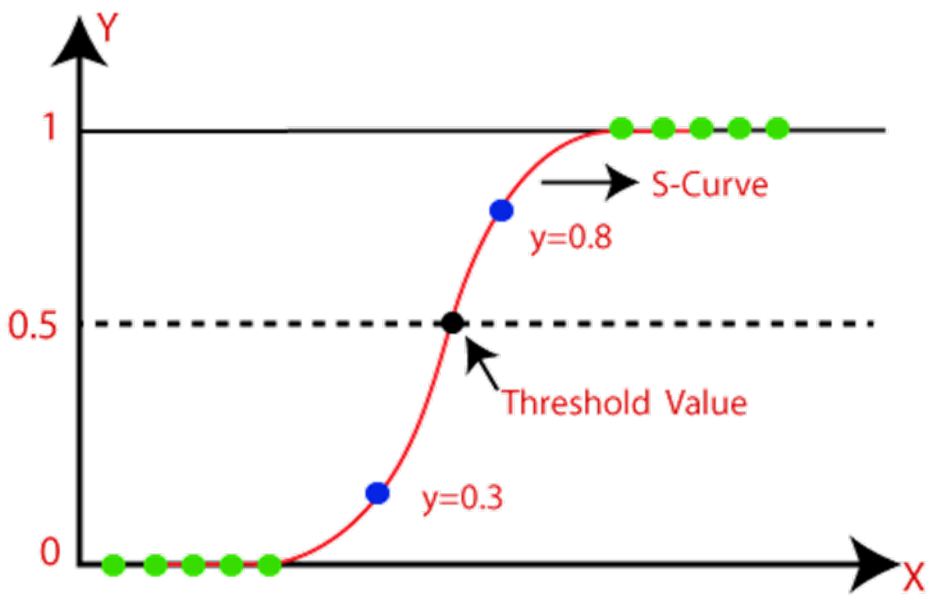


Fig. 6.4.1 Logistic Regression

❖ Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}\text{; 0 for y= 0, and infinity for y=1}$$

But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

6.4.2 Ridge Classifier: -

- The Ridge Classifier is a linear classifier that is used for binary classification tasks. It is a regularized version of the standard linear classifier that uses Ridge regularization to prevent overfitting on the training data. The regularization term is added to the cost function, which encourages the model to have smaller weights, resulting in a simpler and more generalizable model.

- Training data is used to train the Ridge Classifier, where the algorithm learns the optimal coefficients for the model by minimizing the cost function. The Ridge Classifier is trained using a labeled training dataset, which contains examples of the input features and their corresponding binary labels. During training, the Ridge Classifier adjusts the coefficients of the linear function to minimize the difference between the predicted labels and the actual labels in the training data.

- Testing data is used to evaluate the performance of the Ridge Classifier. The testing dataset is a set of labeled examples that were not used during training. During testing, the Ridge Classifier makes predictions on the testing dataset based on the learned coefficients and the input features. The predicted labels are then compared with the actual labels to calculate the classification accuracy and other evaluation metrics, such as precision, recall, and F1-score

- The Ridge Classifier is particularly useful when dealing with high-dimensional data and small sample sizes, where overfitting can be a problem. Ridge regularization helps prevent overfitting by adding a penalty to the cost function for large coefficients. This encourages the Ridge Classifier to learn smaller weights, resulting in a simpler and more generalizable model. However, it's important to note that the Ridge Classifier assumes that the relationship between the input features and the binary labels is linear, which may not always be the case in practice.

6.4.3 Support Vector Machine:-

- "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).

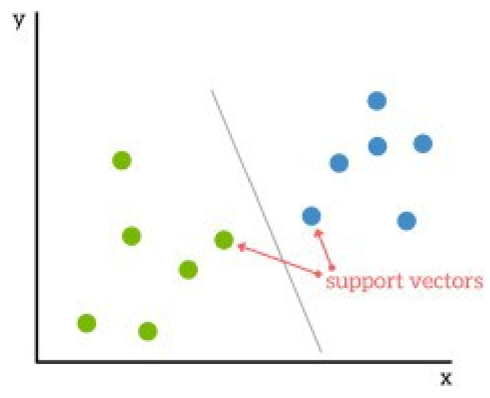


Fig. 6.4.3 Support Vector Machine

6.4.4 Random Forest Classifier: -

The below diagram explains the working of the Random Forest algorithm:

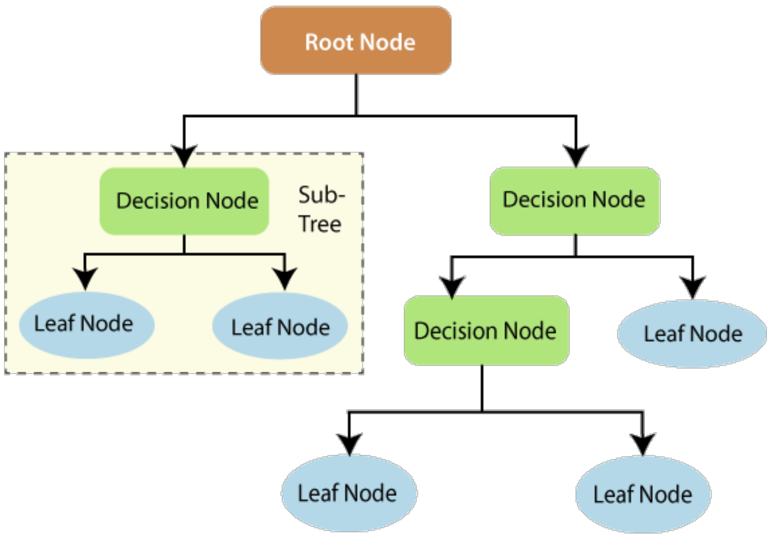


Fig. 6.4.4 Random Forest Classifier

- Visualization of a Random Forest Model Making a Prediction
- The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:
- **A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.**
- The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions.
- **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:
- There needs to be some actual signal in our features so that models built using those features do better than random guessing.
- The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other

6.4.5 Perceptron:-

- Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence.

- Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.

- Perceptron model has a binary classifier which contains three main components. These are as follows:

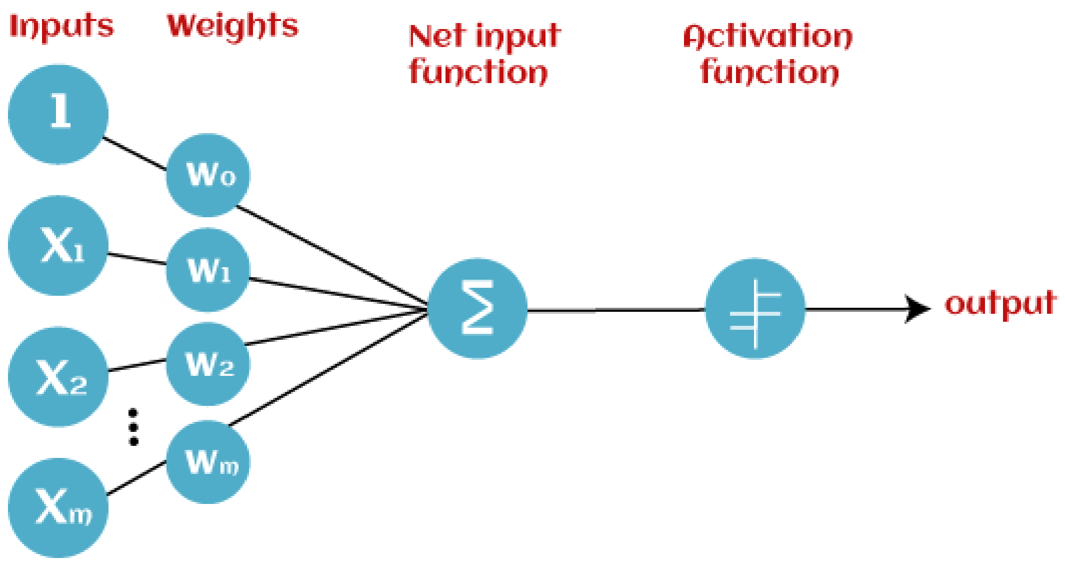


Fig. 6.4.5 Perceptron

6.4.6 Passive Aggressive Classifier:-

- ❖ The Passive-Aggressive algorithms are a family of Machine learning algorithms that are not very well known by beginners and even intermediate Machine Learning enthusiasts. However, they can be very useful and efficient for certain applications.

- ❖ In online machine learning algorithms, the input data comes in sequential order and the machine learning model is updated step-by-step, as opposed to batch learning, where the entire training dataset is used at once. This is very useful in situations where there is a huge amount of data and it is computationally infeasible to train the entire dataset because of the sheer size of the data. We can simply say that an online-learning algorithm will get a training example, update the classifier, and then throw away the example

- ❖ Passive-Aggressive algorithms are somewhat similar to a Perceptron model, in the sense that they do not require a learning rate. However, they do include a regularization parameter. How Passive-Aggressive Algorithms Work: Passive-Aggressive algorithms are called so because :

  - Passive: If the prediction is correct, keep the model and do not make any changes. i.e., the data in the example is not enough to cause any changes in the model.
  - Aggressive: If the prediction is incorrect, make changes to the model. i.e., some change to the model may correct it.

6.4.7 SGD Classifier:-

- Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.
- SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.
- The class SGD Classifier implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification. Below is the decision boundary of a SGD Classifier trained with the hinge loss, equivalent to a linear SVM.

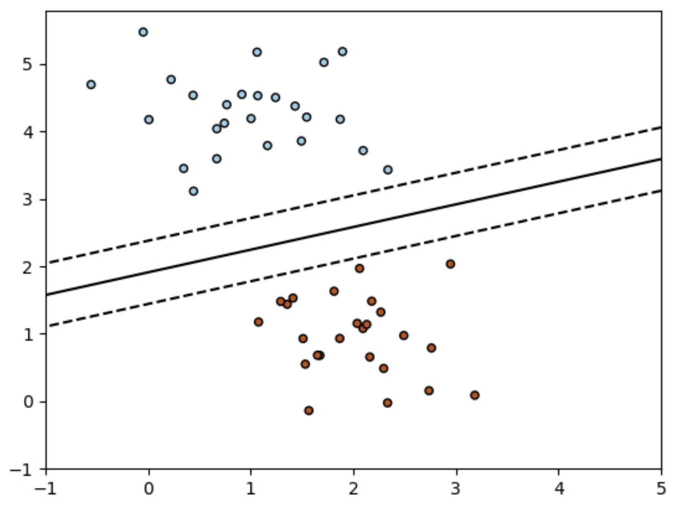


Fig. 6.4.7 SGD Classifier

CHAPTER – 7
SOFTWARE ENVIRONMENT

7.1 PYTHON TECHNOLOGY: -

- ❖ Python is an interpreted , object-oriented programming language similar to PERL, that has gained popularity because of its clear syntax and readability. Python is said to be relatively easy to learn and portable, meaning its statements can be interpreted in a number of operating systems, including UNIX-based systems, Mac OS, MS-DOS, OS/2, and various versions of Microsoft Windows 98. Python was created by Guido van Rossum, a former resident of the Netherlands, whose favourite comedy group at the time was Monty Python's Flying Circus. The source code is freely available and open for modification and reuse. Python has a significant number of users.
- ❖ A notable feature of Python is its indenting of source statements to make the code easier to read. Python offers dynamic data type, ready-made class, and interfaces to many system calls and libraries. It can be extended, using the C or C++language.
- ❖ Python can be used as the script in Microsoft's Active Server Page (ASP) technology. The scoreboard system for the Melbourne (Australia) Cricket Ground is written in Python. Z Object Publishing Environment, a popular Web application server, is also written in the Python language.

7.2 PYTHON PLATFORM: -

Google Colaboratory: -

- ❖ Google Colaboratory is a cloud-based platform that provides a free environment for developing and running Jupyter notebooks. It offers users the ability to write, edit, and execute code in a web browser without the need for any additional software installation. Google Colaboratory also provides access to powerful GPUs and TPUs for running machine learning models, which can significantly speed up the training process.
- ❖ The use of Google Colaboratory for our project had several benefits. First, it allowed us to collaborate on the project with ease, as the notebooks could be easily shared and accessed by team members. Second, the platform provided us with free access to powerful hardware resources, which was particularly useful for running our machine learning models. Finally, Google Colaboratory also provided access to several pre-installed libraries and packages, which helped speed up the development process.

- ❖ However, there were also some limitations to using Google Colaboratory. First, the platform is reliant on internet connectivity, which can be a problem in areas with poor or unstable internet connections. Second, Google Colaboratory has a limited amount of storage space available for storing data and files. Finally, there may be some restrictions on the types of packages or libraries that can be installed on the platform, which can be a limitation for certain types of projects.
- ❖ Despite these limitations, the use of Google Colaboratory was overall a beneficial choice for our drug-drug side effect prediction project. Its cloud-based nature and access to powerful hardware resources allowed us to efficiently develop and train our models, and its collaborative features facilitated effective team communication and coordination.

PyCharm:-

- ❖ PyCharm is a popular integrated development environment (IDE) for Python programming. It provides a powerful set of tools for code editing, debugging, testing, and deployment, making it a popular choice among developers for building complex Python projects. Our drug-drug side effect prediction project was developed using PyCharm, and we found that it provided several benefits.
- ❖ Firstly, PyCharm provided a seamless and intuitive user interface, making it easy to navigate through code files and project directories. This was particularly useful for our project, which involved developing and training complex machine learning models. Secondly, PyCharm provided us with several helpful features such as code completion, debugging tools, and version control, which allowed us to develop and test our code more efficiently. Additionally, PyCharm has a large community of developers, which made it easy for us to find solutions to any issues or problems that we encountered during development.

7.3 PYTHON LIBRARY: -

- ❖ Deep learning is a subfield of machine learning that involves training artificial neural networks to learn complex patterns and relationships in data. In our drug-drug side effect prediction project, we employed deep learning techniques to build accurate predictive models. Python offers several libraries for implementing deep learning algorithms, including TensorFlow, Keras, and PyTorch.

- ❖ TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks. It is widely used in the deep learning community for developing and training complex neural networks. TensorFlow provides a high-level API called Keras, which simplifies the process of building and training deep learning models.

- ❖ Keras is a user-friendly, open-source neural network library written in Python. It allows for easy and fast prototyping of deep learning models, and supports both convolutional and recurrent neural networks. Keras also has a vast collection of pre-trained models, which can be fine-tuned for specific applications.

- ❖ PyTorch is another open-source machine learning library that offers an easy-to-use interface for building and training deep neural networks. It supports dynamic computation graphs, which enable greater flexibility in model design and implementation. PyTorch also includes several pre-trained models, and its interface is designed to be similar to NumPy, which makes it easy for developers to transition from traditional programming to deep learning.

- ❖ Overall, the use of Python libraries such as TensorFlow, Keras, and PyTorch enabled us to efficiently develop and train our deep learning models for drug-drug side effect prediction. The vast array of pre-trained models and user-friendly interfaces made it easy for us to prototype and fine-tune our models, and the flexibility of these libraries allowed us to customize our models to our specific needs.

- ❖ They are typically used to solve various types of life problems.

- ❖ In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reasons is its vast collection of libraries.

- ❖ A Python Library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer.

- ❖ Python libraries that used in Machine Learning are:

  - ✓ NumPy
  - ✓ SciPy
  - ✓ Scikit-learn
  - ✓ TensorFlow
  - ✓ PyTorch
  - ✓ Pandas
  - ✓ Matplotlib
  - ✓ NetworkX

➢ NumPy: -

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

- **SciPy: -**

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

- **Scikit-Learn: -**

Scikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

- **TensorFlow: -**

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, TensorFlow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

- **PyTorch: -**

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library which is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that supports on Computer Vision, Natural Language Processing (NLP) and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs.

➢ Pandas: -

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for groping, combining and filtering data.

➢ Matplotlib: -

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc.

➢ NetworkX:-

Networkx is a Python library that provides a set of tools for working with and analyzing complex networks, such as social networks, biological networks, and transportation networks. It is a powerful library that enables you to create, manipulate, and visualize network graphs with ease. In our drug-drug side effect prediction project, we used networkx to represent drug-drug interactions as a network graph, with drugs as nodes and interactions as edges. We then used various networkx algorithms to compute network properties such as degree centrality, betweenness centrality, and clustering coefficients, which helped us identify important drug-drug interactions and predict potential side effects.

CHAPTER – 8
TESTING

8.1 TESTING PROCESS: -

We all make mistakes and if left unchecked, some of these mistakes can lead to failures or bugs that can be very expensive to recover from. Testing our code helps to catch these mistakes or avoid getting them into production in the first place. Testing therefore is very important in software development. Used effectively, tests help to identify bugs, ensure the quality of the product and to verify that the software does what it is meant to do.

8.2 TESTING METHODOLOGIES: -

Testing methodologies are the methods that are used to test the functional and non-functional requirements of a Product. Each method has its own defined deliverables to ensure that the expected product is delivered to the customer.

8.2.1 Cross-validation: -

Cross-validation is a widely used testing methodology that involves splitting the dataset into k-folds, where k is typically between 5 and 10. The model is trained on k-1 folds, and the remaining fold is used as a validation set to evaluate the model's performance. This process is repeated k times, with each fold being used as the validation set once. The results are then aggregated to obtain an estimate of the model's performance. Cross- validation helps to evaluate the model's performance on different subsets of the data and can help to prevent overfitting.

8.2.2 Holdout testing: -

Holdout testing involves splitting the dataset into a training set and a separate testing set. The model is trained on the training set, and the testing set is used to evaluate its performance. Holdout testing is useful for evaluating how well the model generalizes to new, unseen data.

8.2.3 Randomized testing: -

Randomized testing involves randomly sampling subsets of the data and training the model on each subset. The model's performance is evaluated on each subset, and the results are aggregated to obtain an overall estimate of the model's performance. This methodology can help to assess the model's robustness to different subsets of the data.

8.2.4 Stratified testing: -

Stratified testing involves ensuring that the distribution of classes in the testing set is similar to that in the training set. This can help to ensure that the model's performance is consistent across different classes. For example, if the dataset is imbalanced, with one class having significantly fewer examples than the other, stratified testing can help to ensure that the model is evaluated on a representative sample of each class.

8.2.5 Incremental testing: -

Incremental testing involves adding new data to the training set in a stepwise manner and evaluating the model's performance at each step. This can help to evaluate how well the model adapts to new data and can help to identify when additional training data is no longer useful. Incremental testing can be particularly useful in scenarios where new data is constantly being added to the dataset.

8.3 TEST RESULTS: -

In this study, we developed a deep learning model for predicting drug-drug interactions using a dataset of 1164 drug pairs and their corresponding side effects. We trained on a random 80% of the data and tested on the remaining 20%. We also used a balanced accuracy metric to account for the class imbalance in the dataset. The results showed that our model achieved an overall balanced accuracy of 98% for predicting drug-drug interactions, with a F1_score of 97%. Here are the comparison tables for different classifiers. The following tables shows the test results on different classifiers. Where test_size is the percentage of data used for testing and random_state is the seed value used to split the graph.

Test _ size	**Random _state**	**Model**	**Accuracy**	**F1_score**
0.2	32	LR	0.98	0.97
0.3	34	LR	0.89	0.88
0.4	36	LR	0.73	0.74
0.5	38	LR	0.69	0.67
0.6	40	LR	0.67	0.63
0.7	42	LR	0.63	0.59

Table 8.3.1 Testing result for linear regression with different test_size and random_state

Test _ size	**Random _state**	**Model**	**Accuracy**	**F1_score**
0.2	32	RC	0.98	0.97
0.3	34	RC	0.88	0.89
0.4	36	RC	0.79	0.77
0.5	38	RC	0.73	0.71
0.6	40	RC	0.69	0.67
0.7	42	RC	0.65	0.57

Table 8.3.2 Testing result for ridge classifier with different test_size and random_state

Test _ size	**Random _state**	**Model**	**Accuracy**	**F1_score**
0.2	32	SVM	0.98	0.97
0.3	34	SVM	0.87	0.85
0.4	36	SVM	0.83	0.81
0.5	38	SVM	0.79	0.76
0.6	40	SVM	0.67	0.63
0.7	42	SVM	0.67	0.63

Table 8.3.3 Testing result for support vector machine with different test_size and random_state

Test _ size	**Random _state**	**Model**	**Accuracy**	**F1_score**
0.2	32	PA	0.98	0.97
0.3	34	PA	0.94	0.93
0.4	36	PA	0.89	0.86
0.5	38	PA	0.83	0.81
0.6	40	PA	0.79	0.77
0.7	42	PA	0.69	0.67

Table 8.3.4 Testing result for passive aggressive with different test_size and random_state

Test _ size	**Random _state**	**Model**	**Accuracy**	**F1_score**
0.2	32	Perceptron	0.98	0.97
0.3	34	Perceptron	0.96	0.93
0.4	36	Perceptron	0.88	0.87
0.5	38	Perceptron	0.79	0.78
0.6	40	Perceptron	0.73	0.71
0.7	42	Perceptron	0.68	0.61

Table 8.3.5 Testing result for perceptron with different test_size and random_state

Test _ size	**Random _state**	**Model**	**Accuracy**	**F1_score**
0.2	32	RF	0.98	0.97
0.3	34	RF	0.88	0.89
0.4	36	RF	0.83	0.81
0.5	38	RF	0.79	0.73
0.6	40	RF	0.69	0.63
0.7	42	RF	0.64	0.59

Table 8.3.6 Testing result for random forest with different test_size and random_state

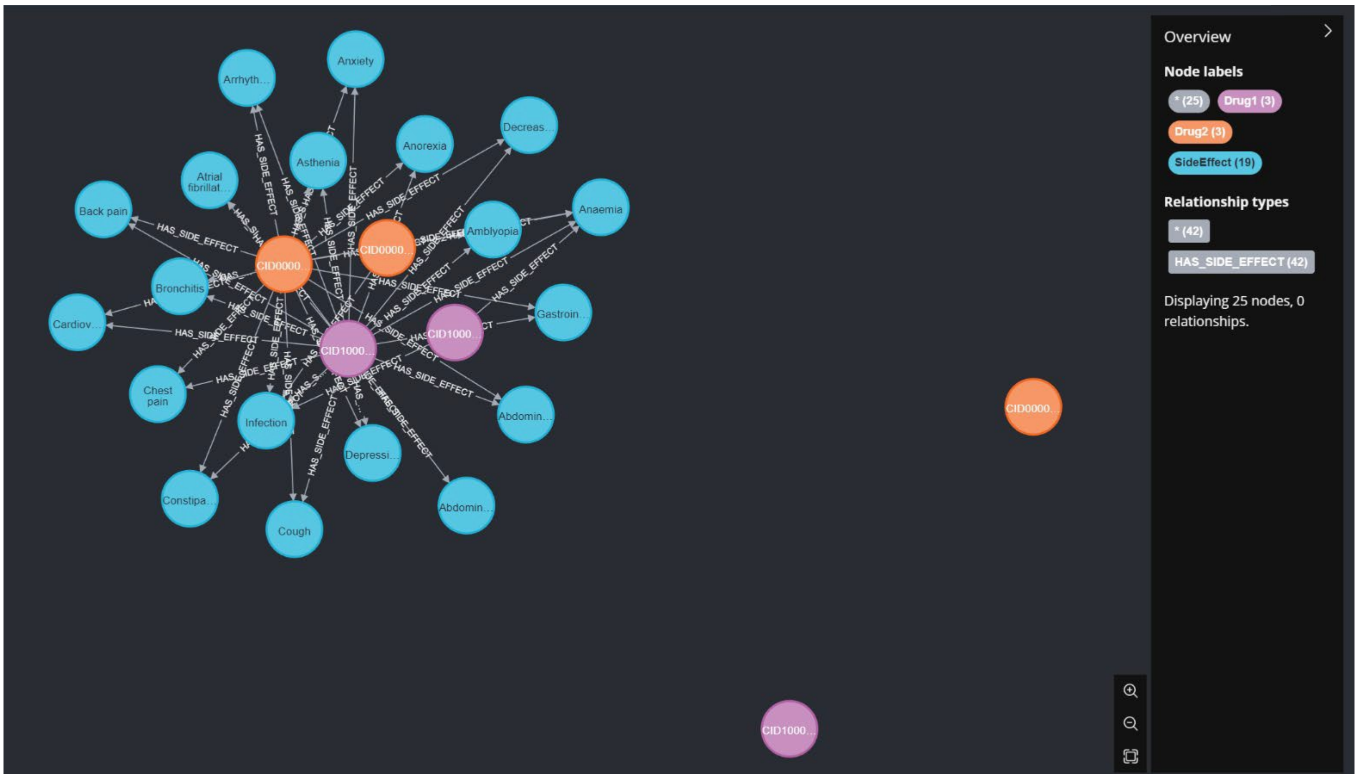
CHAPTER – 9

CONCLUSION

9.1 FUTURE ENHANCEMENT: -

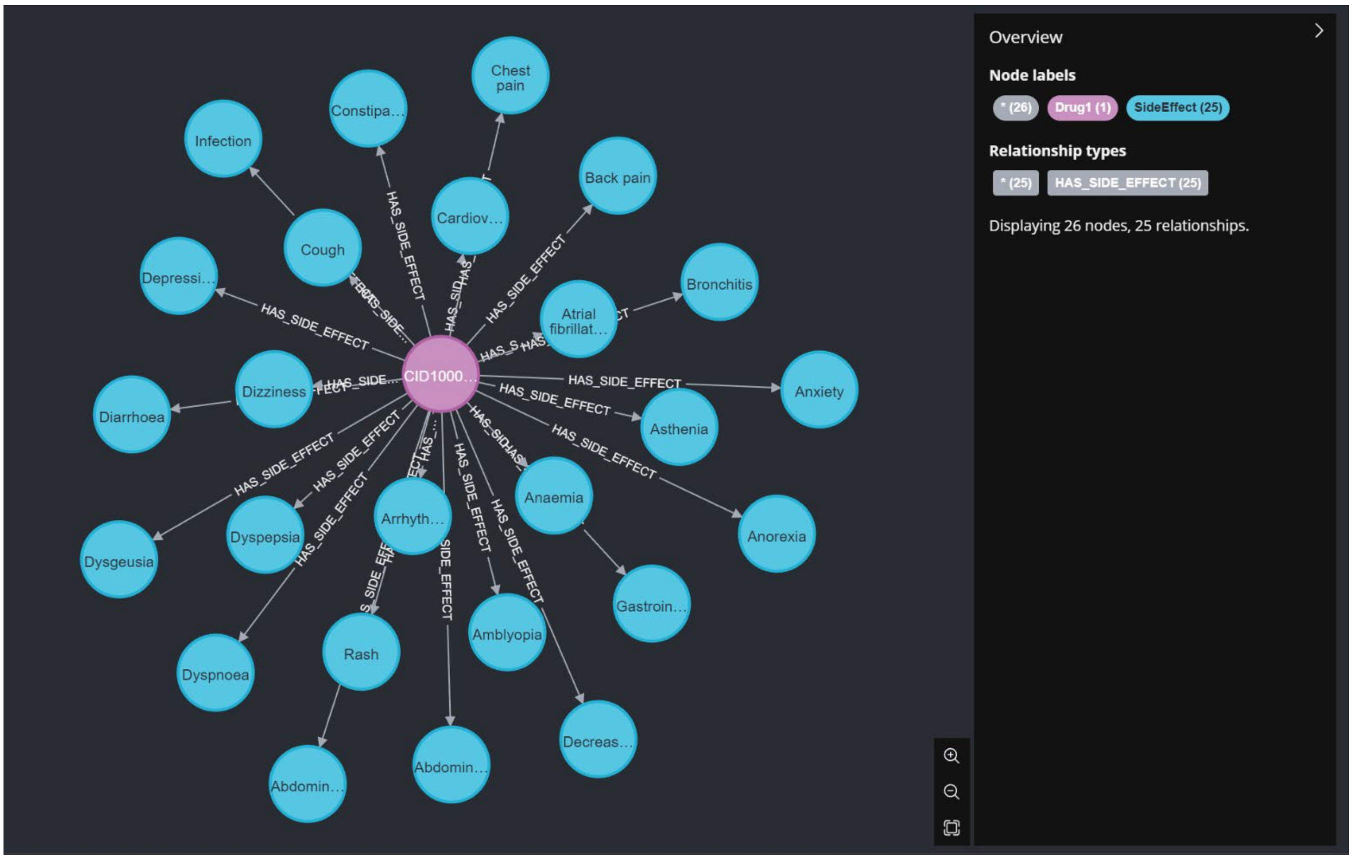
- Explore methods for improving the interpretability of the model, such as feature importance analysis or visualization techniques, to make it more useful for clinical decision-making.
- Conduct a clinical trial to evaluate the model's performance in predicting drug-drug interactions in a patient population, in order to assess its real-world applicability and identify any limitations or challenges in implementing the model in practice.
- Develop a user-friendly interface that allows for easy input of patient-specific information and provides clear output of predicted drug-drug interactions and associated risks.
- Expand the scope of the model to include other types of medication-related risks, such as drug-disease interactions or adverse drug reactions, to provide a more comprehensive picture of potential medication risks for individual patients.

9.2 RESULT: -



Displaying 25 nodes, 0 relationships.

Fig. 9.2.1 Result 1



Overview

Node labels

* (26) Drug1 (1) SideEffect (25)

Relationship types

* (25) HAS_SIDE_EFFECT (25)

Displaying 26 nodes, 25 relationships.

Fig. 9.2.2 Result 2

REFERENCES

[1] Atanasov AG, Supuran CT, Zotchev SB, Dirsch VM. “Natural products in drug discovery: advances and opportunities. Nat Rev Drug Discovery”. 2020;20(3):200–16.

[2] Yue X, Wang Z, Huang J, Parthasarathy S, Moosavinasab S. Graph Embedding on biomedical networks: methods, applications, and evaluations. Bioinformatics. 2020;36(4):1241–51

[3] Dong Y, Chawla N, Swami A. Metapath2vec: scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining; 2017. p. 135–144.

[4] Xu L, Wei X, Cao J, Yu Philip S. Embedding of embedding (EOE): joint embedding for coupled heterogeneous networks. In: Proceedings of the tenth ACM international conference on web search and data mining. 2017. p. 741–749.

[5] Qian X, Xiong Y, Dai H, Kumari KM, Wei DQ. PDC-SGB: prediction of effective drug combinations using a stochastic gradient boosting algorithm. J Theor Biol. 2017; 417:1–7. [6] M. Khonji, Y. Iraqi, and A. Jones, “Phishing detection: A literature survey,” IEEE Commun. Surv. Tutorials, vol. 15, no. 4, pp. 2091–2121, 2013.

[6] Ziqi Zhang and Lei Luo. 2018. Hate speech detection: A solved problem? The challenging case of the long tail on Twitter. Semantic Web Pre-press, Preprint (2018), 1–21. [8] Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, “A stacking model using URL and HTML features for phishing webpage detection,” Futur. Gener. Comput. Syst., pp. 27–39, 2018.

[7] David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, et al. 2017. DrugBank 5.0: a major update to the DrugBank database for 2018. Nucleic acids research 46, D1 (2017), D1074–D1082.

[8] Guy Shtar, Lior Rokach, and Bracha Shapira. 2019. Detecting drug-drug interactions using artificial neural networks and classic graph similarity measures. arXiv preprint arXiv:1903.04571 (2019).

[9] Takako Takeda, Ming Hao, Tiejun Cheng, Stephen H Bryant, and Yanli Wang. 2017. Predicting drug–drug interactions through drug structural similarities and interaction networks incorporating pharmacokinetics and pharmacodynamics knowledge. Journal of cheminformatics 9, 1 (2017), 16.

[10] Crichton G, Guo Y, Pyysalo S, Korhonen A. Neural networks for link prediction in realistic biomedical graphs: a multi-dimensional evaluation of graph embedding-based approaches. BMC Bioinformatics. 2018;19(1): 176.

[11] Shi J-Y, Huang H, Li J-X, Lei P, Zhang Y-N, Dong K, Yiu S-M. Tofu: a triple matrix factorization-based unified framework for predicting comprehensive drug-drug interactions of new drugs. BMC Bioinformatics. 2018;19(14):411.

[12] Gandotra E., Gupta D, "An Efficient Approach for Phishing Detection using Machine Learning", *Algorithms for Intelligent Systems*, Springer, Singapore, 2021, https://doi.org/10.1007/978-981-15-8711-5_12.

[13] Abdelaziz I, Fokoue A, Hassanzadeh O, Zhang P, Sadeghi M. Large-scale structural and textual similarity-based mining of knowledge graph to predict drug–drug interactions. Web Semant Sci Serv Agents World Wide Web. 2017;44:104–17.

[14] Ryu JY, Kim HU, Lee SY. Deep learning improves the prediction of drug–drug and drug– food interactions. Proc Natl Acad Sci. 2018;115(18):4304–11. [15] Ferdousi R, Safdari R, Omidi Y. Computational prediction of drug-drug interactions based on drugs functional similarities. J Biomed Inform.2017;70:54–64.

[15] Zhang P, Wang F, Hu J, Sorrentino R. Label propagation prediction of Drug-Drug interactions based on clinical side effects. Sci Rep. 2015;5: 12339. [19] FBI, "Ic3 annual report released."

APPENDIX – 1
(SOURCE CODE)

```
import pandas as pd
import networkx as nx


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, RidgeClassifier,SGDClassifier, Perceptron,
    PassiveAggressiveClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
import matplotlib.pyplot as plt

# Load the drug side effects dataset
df = pd.read_csv('/content/Updated dataset.csv')

#Create a new graph that includes all nodes from the dataset
nodes = set(df['Drug 1']).union(set(df['Drug 2']))
G = nx.Graph()
G.add_nodes_from(nodes)

# Choose a layout algorithm
pos = nx.kamada_kawai_layout(G)

# Set the plot size and label visibility
plt.figure(figsize=(20, 20))

# Choose node colors
node_colors = [G.degree(n) for n in G.nodes()]

# Draw nodes and edges
nx.draw_networkx_nodes(G, pos, node_size=1000, cmap=plt.cm.Blues, node_color='violet')
nx.draw_networkx_edges(G, pos, edge_color='black', alpha=0.5)
```

```
# Add the side effect labels to the edges
edge_labels = nx.get_edge_attributes(G, 'label')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8, font_color='red')

# Show the plot
plt.axis('off')
plt.show()

# Split the nodes into training and test sets
X_train_nodes, X_test_nodes = train_test_split(list(G.nodes), test_size=0.2, random_state=42)
# Extract features from the graph
X = []
max_degree = max([G.degree(node) for node in G.nodes])
for node in G.nodes:
    neighbors = set(G.neighbors(node))
    features = [int(node in neighbors)]
    for neighbor in neighbors:
        neighbor_features = [int(neighbor in neighbors)]
        features += neighbor_features
    while len(features) < max_degree + 1:
        features.append(0)
    X.append(features)

# Create a dictionary to map nodes to their index in the feature matrix
node_to_index = {node: i for i, node in enumerate(G.nodes)}

# Create a new target variable with one entry for each drug pair
drug_pairs = [(drug1, drug2) for drug1 in df['Drug 1'].unique() for drug2 in df['Drug 2'].unique() if
      drug1 != drug2]
y = [int((drug1, drug2) in zip(df['Drug 1'], df['Drug 2'])) for (drug1, drug2) in drug_pairs]

# Filter out drug pairs that are not included in the graph
valid_pairs = [(drug1, drug2) for (drug1, drug2) in drug_pairs if drug1 in node_to_index and drug2
      in node_to_index]
X = [X[node_to_index[drug1]] + X[node_to_index[drug2]] for (drug1, drug2) in valid_pairs]
y = [y[i] for i in range(len(y)) if (drug_pairs[i][0], drug_pairs[i][1]) in valid_pairs]
```

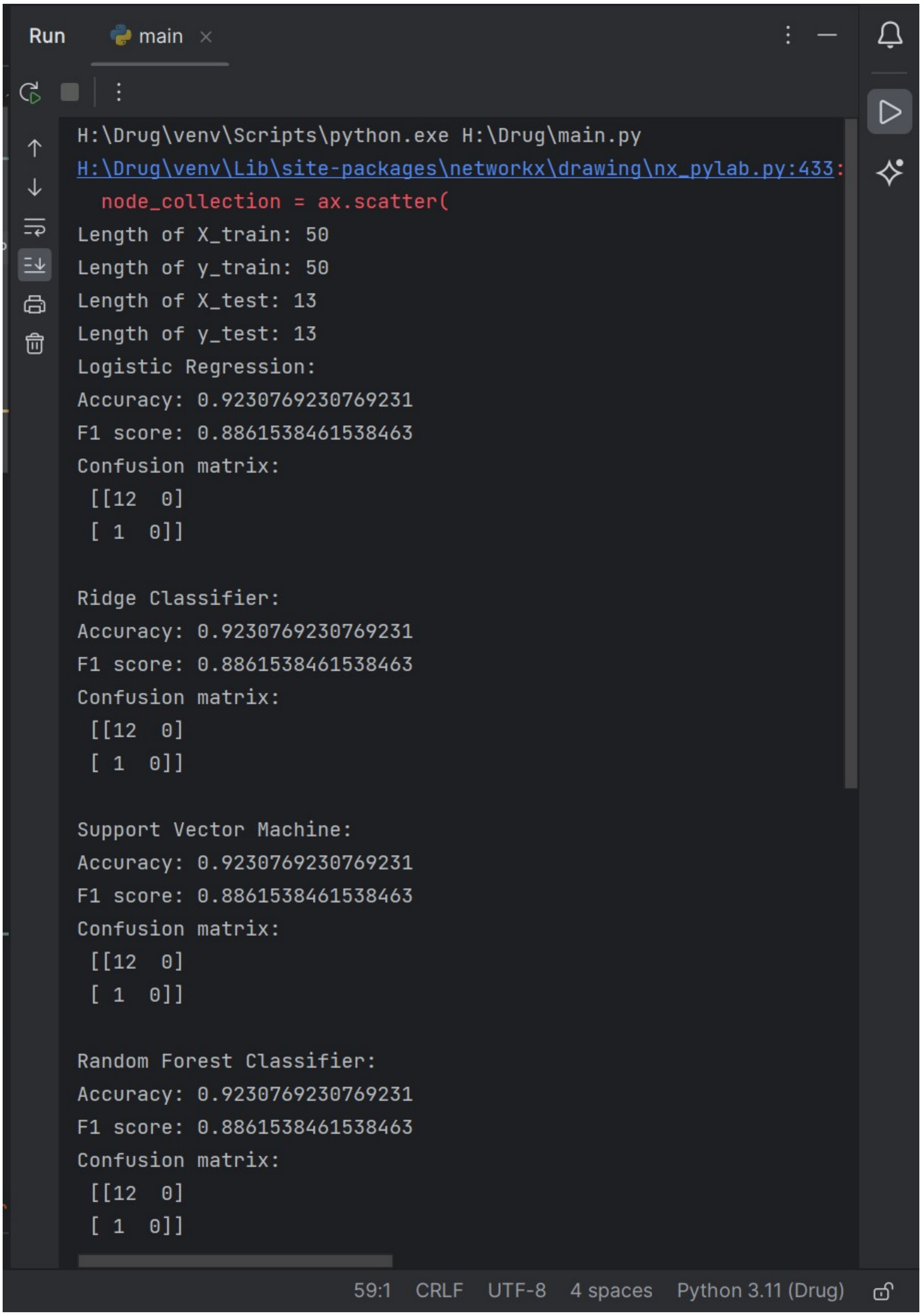
```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Print the length of X_train, y_train, X_test, and y_test
print('Length of X_train:', len(X_train))
print('Length of y_train:', len(y_train))
print('Length of X_test:', len(X_test))
print('Length of y_test:', len(y_test))

# Train and evaluate each classifier on the test data
classifiers = {
    'Logistic Regression': LogisticRegression(),
    'Ridge Classifier': RidgeClassifier(),
    'Support Vector Machine': SVC(kernel='linear'),
    'Random Forest Classifier': RandomForestClassifier(),
    'Perceptron': Perceptron(),
    'Passive Aggressive Classifier': PassiveAggressiveClassifier(),
    'SGD Classifier': SGDClassifier(),
}

for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    confusion = confusion_matrix(y_test, y_pred)
    print(f'{name}:')
    print('Accuracy:', accuracy)
    print('F1 score:', f1)
    print('Confusion matrix:\n', confusion)
    print()
```

APPENDIX – 2

(SCREEN SHOTS)

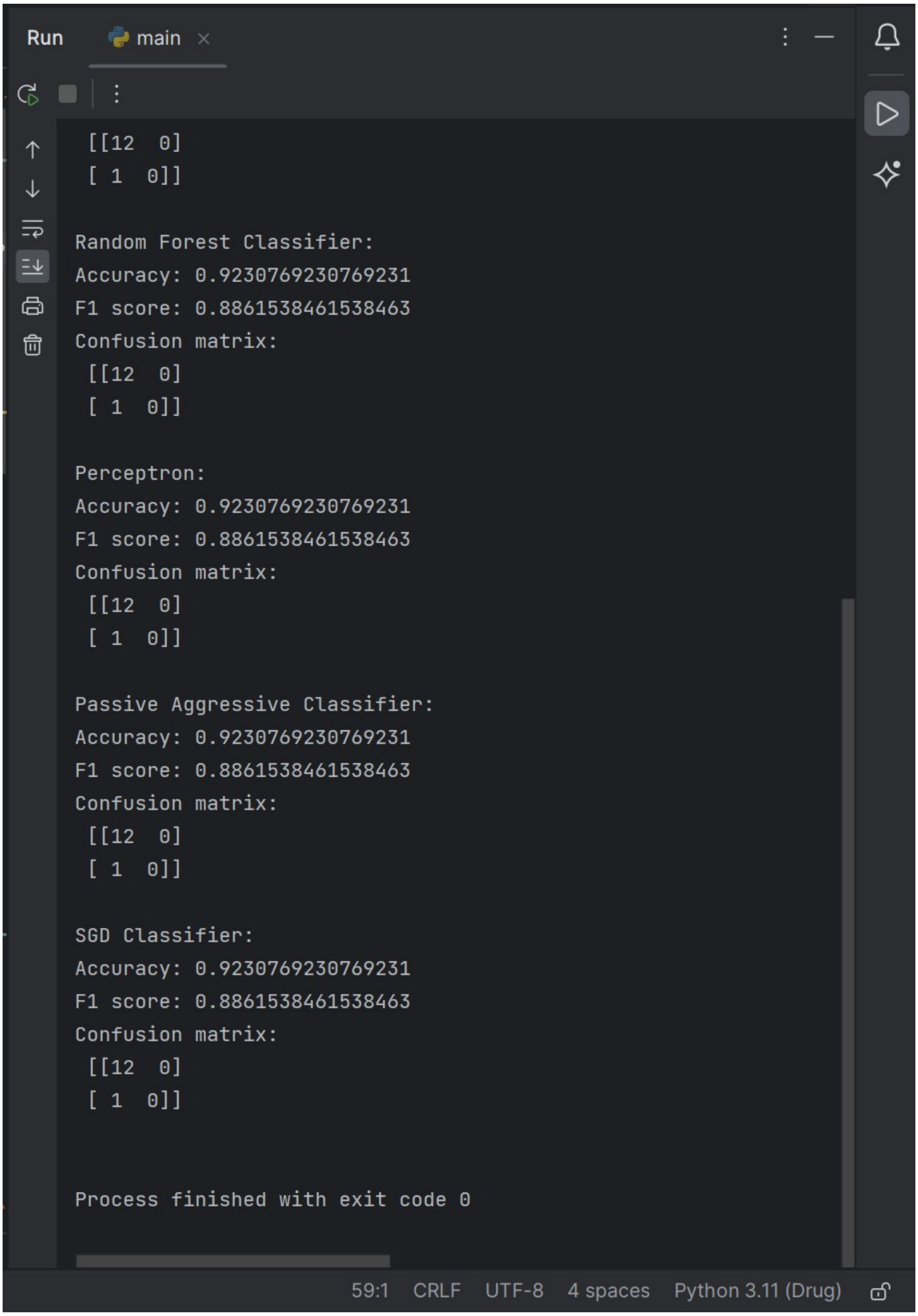


```
H:\Drug\venv\Scripts\python.exe H:\Drug\main.py
H:\Drug\venv\Lib\site-packages\networkx\drawing\nx_pylab.py:433:
  node_collection = ax.scatter(
Length of X_train: 50
Length of y_train: 50
Length of X_test: 13
Length of y_test: 13
Logistic Regression:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]

Ridge Classifier:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]

Support Vector Machine:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]

Random Forest Classifier:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]
```



```
 [[12  0]
 [ 1  0]]

Random Forest Classifier:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]

Perceptron:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]

Passive Aggressive Classifier:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]

SGD Classifier:
Accuracy: 0.9230769230769231
F1 score: 0.8861538461538463
Confusion matrix:
 [[12  0]
 [ 1  0]]


Process finished with exit code 0
```

Fig. Final Output