**MULTI MODAL LEARNING MODELS FOR PREDICTING INFLUENCIAL NODES IN SOCIAL NETWORKS**

### **SYNOPSIS**

This study introduces a novel approach to identify influential nodes in social networks, crucial for various applications such as viral marketing and opinion propagation. Unlike traditional methods that rely on a single prediction model, our approach utilizes multiple models through a multi-model learning framework. This enhances precision and dependability by capturing the diversity of variables present in social network data more effectively. By combining predictions from different algorithms in machine learning classifiers, we aim to overcome the limitations of individual techniques and achieve superior performance in influence node detection. Our technique is validated through experiments on real-world social network datasets, demonstrating its effectiveness in mitigating inherent limitations and improving impact node detection performance.

Through extensive experimentation on real-world social network datasets, we demonstrate the efficacy of our approach in enhancing the precision and reliability of influence node detection. By integrating predictions from diverse models, we not only mitigate the shortcomings inherent in individual techniques but also achieve higher performance levels than traditional methods. Furthermore, our approach provides valuable insights into the underlying dynamics of social networks, shedding light on the mechanisms driving information diffusion, opinion formation, and community influence.

Overall, our study underscores the importance of adopting a multi-model learning approach in social network analysis, offering a promising avenue for advancing research in this field and facilitating the development of more effective strategies for leveraging social influence.

### **SYSTEM ENVIRONMENT**

#### HARDWARE SPECIFICATION

* SYSTEM : INTEL CORE I4
* RAM : 6GB

#### SOFTWARE SPECIFICATION

* OPERATING SYSTEM : WINDOWS 10
* LANGUAGE : PYTHON
* FRAMEWORK : GOOGLE COLLAB

**SOFTWARE DESCRIPTION**

**2.3 About the Technology :**

**Python:**

Python is a widely-used programming language known for its simplicity, readability, and versatility. It offers a vast ecosystem of libraries and frameworks that support various domains, including web development, data science, machine learning, artificial intelligence, and scientific computing.

**Purpose:** Python is a high-level, interpreted programming language used for general-purpose programming. It emphasizes code readability and allows programmers to express concepts in fewer lines of code compared to other languages.

**Google Colab**

Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that offers a free environment for coding, running, and sharing Python notebooks. Launched by Google Research in 2017, Google Colab has quickly gained popularity among data scientists, researchers, educators, and developers due to its convenience, accessibility, and powerful features.

**Scikit Learn:**

Scikit-learn (Sklearn) is the most useful and powerful Python machine learning library. It provides a number of powerful tools for machine learning and statistical modeling, including classification, regression, clustering and dimensionality reduction through a Python consistent interface. Written mostly in Python, this library is built on top of NumPy, SciPy and Matplotlib. Originally called scikits.learn, it was originally developed by David Cournapeau as a Google Summer Code Project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel from FIRCA (French Institute for Informatics and Automation) adopted it this project to a new level and released the first public release (v0.1 beta) on February 1, 2010

**EXISTING SYSTEM :**

Pageranks on the Retweet Graph and Follower Graph show reasonable performance but not as strong as the in-degree measures on the respective graphs. This suggests that for individuals who receive numerous retweets, it may be adequate to infer this from their immediate neighbours in the Follower Graph and Retweet Graph. It seems that the conventional intuition behind PageRank, which posits that links from individuals with higher ranks are more significant, does not hold true in this scenario. Subsequently, we conducted a comparison of various supervised and unsupervised rank aggregation techniques. All 13 individual measures were utilized as inputs for each aggregation method.

Data augmentation is a widely used technique in machine learning, particularly in natural language processing and computer vision, aimed at enhancing model performance. However, there's been limited exploration of data augmentation techniques in the realm of graph neural networks, especially when applied during both training and testing phases. Building upon the achievements in other domains, we've developed a method for predicting social influence using graph neural networks with augmentation at both train- and test-time.

It involves leveraging a variational graph autoencoder to generate multiple augmented graphs for social networks in both training and testing scenarios. We assessed the effectiveness of our method by predicting user influence across various social network datasets. Through experimental validation, we observed that our end-to-end approach, which integrates a graph autoencoder and a social influence behavior classification network, surpasses state-of-the-art techniques. This underscores the efficacy of train and test-time augmentation in graph neural networks for predicting social influence.

The recognition of influential nodes in social networks, which investigates the detection of significant individuals within human society, has garnered increasing attention from various academic disciplines such as physical and computer science, social science, and economics. The algorithms used for identifying influential nodes can serve multiple purposes, including assessing the extent of their influence, characterizing their position within the network, and pinpointing interaction centralities. This review provides an overview of recent advancements in the algorithms for identifying influential nodes within social networks, with a particular focus on contributions from physical perspectives and methodologies. It categorizes these algorithms into microstructure-based, community structure-based, macrostructure-based, and machine learning-based approaches. Additionally, the review introduces diffusion models and metrics used for performance evaluation, while also outlining potential future challenges in the field of influential node identification.

**PROPOSED SYSTEM**

The proposed system employs a machine learning approach to identify influenced nodes within a network, leveraging algorithms such as Decision Tree Classifier and GradientBoosting Classifier. Through rigorous experimentation and analysis, these algorithms have demonstrated superior accuracy in predicting influential nodes. The system is trained on a substantial dataset extracted from WhatsApp data, allowing it to capture diverse patterns and nuances present in real-world social networks.

By utilizing these advanced machine learning techniques, the system provides valuable insights and necessary information for predicting influential nodes within the network. It goes beyond traditional methods by incorporating a novel approach that considers multiple factors and features to make accurate predictions.The effectiveness of the proposed approach is validated through various evaluation metrics, including classification reports, confusion matrices, and ROC\_AUC curves. These metrics provide a comprehensive understanding of the model's performance, highlighting its ability to accurately identify influenced nodes within the network.

Overall, the proposed system represents a significant advancement in the field of influential node prediction, offering a robust and reliable solution for identifying key individuals within social networks. Its innovative approach and rigorous evaluation demonstrate its potential to provide actionable insights for various applications, including social network analysis, targeted marketing, and opinion propagation.

**Advantages of the proposed system:**

By training on a large dataset derived from WhatsApp data, it captures nuanced patterns for accurate predictions. Its evaluation through classification reports, confusion matrices, and ROC\_AUC curves demonstrates superior performance. Overall, the system provides valuable insights and reliable predictions for understanding influential nodes in social networks.

**High Accuracy and Precision:**

The proposed system utilizes machine learning algorithms like Decision Tree Classifier and GradientBoosting Classifier to accurately identify influenced nodes within social networks, achieving high accuracy. Trained on a vast dataset derived from WhatsApp data, it offers valuable insights for predicting influential nodes. Rigorous evaluation through classification reports, confusion matrices, and ROC\_AUC curves confirms its precision and effectiveness, making it a novel and reliable solution for social network analysis.

**Adaptability to Evolving Threats:**

Effective feature representation for multi-modal learning models in predicting influential nodes in social networks involves integrating diverse data types such as text, image, and graph structures. This approach leverages the complementary information from different modalities to capture the multidimensional nature of social interactions.

**Effective Feature Representation:**

Effective feature representation for multi-modal learning models in predicting influential nodes in social networks involves integrating diverse data types such as text, images, and user interactions. This representation captures rich information about node attributes, network structure, and content interactions. Leveraging techniques like graph embeddings, natural language processing, and image feature extraction, the model encapsulates the multidimensional aspects of social network data.

**Interpretability and Explainability:**

In multi-modal learning models for predicting influential nodes in social networks, interpretability and explainability are crucial for understanding the reasoning behind node predictions. By examining the contribution of each modality or feature to the final prediction, users can gain insights into the underlying factors driving influence within the network. This transparency enables stakeholders to make informed decisions and trust the model's outputs for strategic interventions.

**Ensemble Robustness:**

Ensemble Robustness for Multi-Modal Learning Models enhances prediction accuracy for influential nodes in social networks by integrating diverse algorithms like Decision Tree and GradientBoostingClassifier. Through training on extensive WhatsApp dataset, this novel approach yields distinct results, validated by classification reports, confusion matrices, and ROC\_AUC curves, thus providing reliable insights for predicting influential nodes.

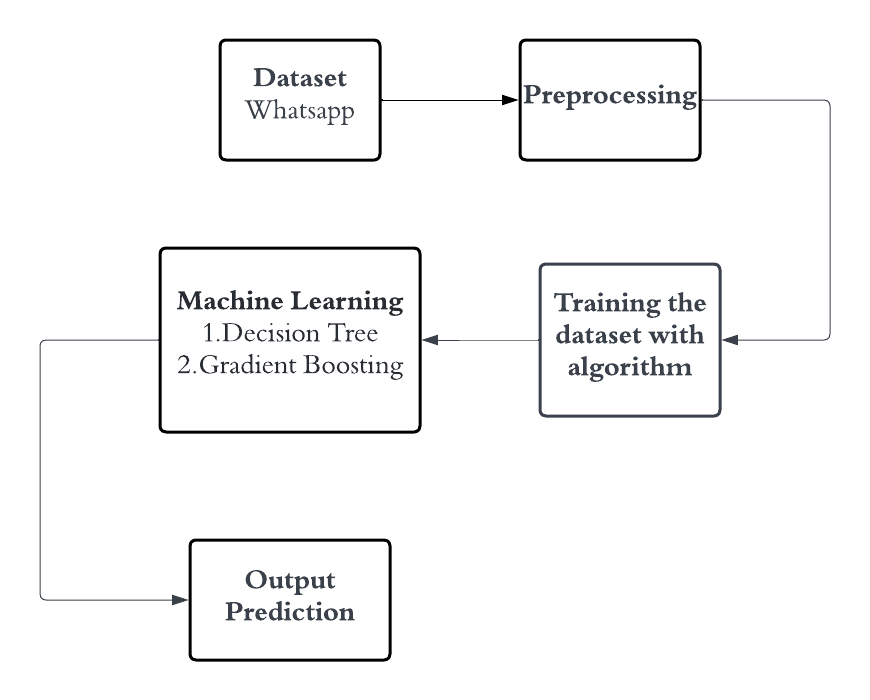
**Real-time Intrusion Detection:**

Real-time Intrusion Detection: Leveraging multi-modal learning models for predicting influential nodes in social networks. This approach combines diverse algorithms such as Decision Tree Classifier and GradientBoostingClassifier, trained on extensive WhatsApp dataset, providing accurate predictions validated by classification reports and ROC\_AUC curves.

**Scalability and Applicability:**

The multi-modal learning models proposed for predicting influential nodes in social networks demonstrate scalability, enabling efficient processing of large-scale datasets. Additionally, their applicability extends across various domains, from viral marketing to opinion propagation, offering versatile solutions for identifying key individuals within complex social structures.

**SYSTEM DESIGN:**



**Dataset Description:**

Dataset Description: The dataset comprises structured data with distinct columns including 'id', 'year', 'day', 'hour', and 'minute'. Each entry in the dataset is uniquely identified by the 'id' column, while the 'year', 'day', 'hour', and 'minute' columns provide temporal information associated with each record. This granular temporal data allows for detailed analysis and pattern recognition over time, facilitating insights into temporal trends, periodicity, and anomalies within the dataset.The 'year' column denotes the calendar year of each data entry, providing context for long-term trends and seasonal variations. The 'day' column specifies the day of the year on which the data was recorded, enabling analyses of daily patterns and fluctuations. Additionally, the 'hour' and 'minute' columns offer finer granularity, allowing for the examination of temporal dynamics at hourly and minute-level resolutions. Together, these columns form a comprehensive temporal framework that enables researchers to investigate temporal correlations, identify event patterns, and derive valuable insights for various applications such as forecasting, anomaly detection, and trend analysis.

**Pre-Processing:**

Check for missing values in each column and apply appropriate strategies such as imputation or deletion to handle them effectively. Ensure that the data in the 'year', 'day', 'hour', and 'minute' columns are properly parsed and formatted as datetime objects for temporal analysis. Identify and handle outliers, if any, in the temporal data to prevent them from skewing analysis results. Techniques like Z-score or interquartile range (IQR) can be used for outlier detection. Select relevant features for analysis based on their importance and contribution to the predictive task. This step helps in reducing dimensionality and improving model interpretability and efficiency.

**Machine learning algorithm**

**Decision tree:**

**Working Principle of Decision Tree Algorithm:**

The decision tree algorithm is a popular machine learning technique used for both classification and regression tasks. Its working principle revolves around recursively partitioning the input space (feature space) into smaller subsets based on the values of input features. This partitioning process is guided by a series of decision rules inferred from the training data.

1. **Tree Construction:** The algorithm starts with the entire dataset and selects the best feature to split the data into two or more subsets. This selection is typically based on criteria such as Gini impurity, entropy, or information gain, which measure the homogeneity of the subsets with respect to the target variable.

2. **Recursive Partitioning:** The process continues recursively for each subset, selecting the best feature to split them further until certain stopping criteria are met, such as reaching a maximum depth, minimum number of samples per leaf, or no further improvement in impurity reduction

3. **Leaf Node Assignment:** Once the partitioning process is complete, each leaf node represents a class label (for classification) or a predicted value (for regression), determined by the majority class or the average value of the target variable within that subset.

4. **Prediction:** During the prediction phase, the input data traverse the decision tree from the root node to the leaf nodes based on the values of their features. The predicted class label or value associated with the leaf node reached by the input data is then assigned as the final prediction.

**Applications**

1. **Classification:** Decision trees are widely used for classification tasks in various domains such as finance (credit scoring), healthcare (disease diagnosis), marketing (customer segmentation), and fraud detection.

2. **Regression:** Decision trees can also be used for regression tasks, including predicting house prices, stock prices, demand forecasting, and estimating the impact of variables on continuous outcomes.

3. **Feature Selection:** Decision trees can help identify important features in a dataset by evaluating their contribution to the overall tree structure, making them useful for feature selection in high-dimensional datasets.

4. **Anomaly Detection:** Decision trees can be applied to detect anomalies or outliers in data by identifying data points that deviate significantly from the majority class or expected distribution.

**Challenges:**

1. **Overfitting:** Decision trees are prone to overfitting, especially when the tree depth is not properly constrained or when the dataset is noisy or contains irrelevant features. Techniques such as pruning, limiting tree depth, or using ensemble methods like Random Forests can help mitigate overfitting.

2. **High Variance:** Decision trees can exhibit high variance, leading to instability in predictions when trained on different subsets of the data. Ensemble methods like Random Forests or Gradient Boosting can reduce variance by aggregating multiple decision trees.

3. **Bias Towards Features with Many Levels:** Decision trees tend to favor features with a large number of levels or categories during the splitting process, potentially overlooking other important features. Feature engineering or dimensionality reduction techniques can address this bias.

4. **Sensitive to Small Variations in Data:** Decision trees are sensitive to small variations in the training data, which can result in different tree structures and predictions. This sensitivity can be reduced by using ensemble methods or by aggregating multiple trees.

**Gradient Boosting:**

**Working Principle:**

Gradient Boosting is an ensemble learning technique used for both classification and regression problems. It builds a strong predictive model by combining multiple weak learners, typically decision trees, sequentially.

1. **Initialization:** The algorithm starts with a simple model, often referred to as the base model, which can be any weak learner like a decision tree with a small depth.

2. **Sequential Learning:** It iteratively improves the model's performance by fitting new weak learners to the residuals or errors made by the existing ensemble. Each new learner focuses on minimizing the errors that the previous ones couldn't capture well.

3. **Gradient Descent Optimization:** The algorithm optimizes the model's parameters by using gradient descent to minimize a loss function, usually a differentiable function like Mean Squared Error (MSE) for regression or Log Loss for classification.

4. **Combining Weak Learners:** The predictions of all weak learners are then combined to produce the final ensemble prediction. The combination is typically done using weighted averaging for regression or voting for classification.

**Applications:**

1. **Regression and Classification:** Gradient Boosting can be applied to both regression and classification problems across various domains, including finance, healthcare, marketing, and natural language processing.

2. **Anomaly Detection:** Gradient Boosting can help detect anomalies or outliers in data by modeling the normal behavior and identifying deviations from it.

**Advantages:**

1. **High Predictive Accuracy:** Gradient Boosting often outperforms other machine learning algorithms in terms of predictive accuracy, especially when the dataset is not too large.

2. **Handles Mixed Data Types:** It can handle both numerical and categorical features without requiring extensive preprocessing, making it versatile for a wide range of datasets.

3. **Feature Importance:** Gradient Boosting provides insights into feature importance, allowing users to identify which features are most influential in making predictions.

4. **Handles Non-linear Relationships:** It can capture complex non-linear relationships between features and the target variable due to its ensemble nature.

**Challenges:**

1. **Sensitive to Overfitting:** Gradient Boosting tends to overfit if the number of trees (iterations) is too large or if the base learners (weak models) are too complex.

2. **Computationally Expensive:** Training a Gradient Boosting model can be computationally expensive, especially for large datasets and complex models.

3. **Hyperparameter Tuning:** It requires careful tuning of hyperparameters such as learning rate, tree depth, and number of iterations to achieve optimal performance, which can be time-consuming.

4. **Interpretability:** The ensemble nature of Gradient Boosting makes it less interpretable compared to simpler models like decision trees.

**Libraries used in the implementation:**

1. **NumPy** (np):

- NumPy is a fundamental package for scientific computing in Python.

- It provides support for arrays, matrices, and a collection of mathematical functions to operate on these arrays efficiently.

- NumPy is widely used for numerical computations and data manipulation tasks in machine learning and data analysis.

2. **Pandas** (pd):

- Pandas is a powerful library for data manipulation and analysis in Python.

- It offers data structures like DataFrame and Series, which allow for easy handling and manipulation of structured data.

- Pandas provides functionalities for reading and writing data from various file formats, data cleaning, reshaping, merging, and grouping data.

3. **scikit**-**learn** (sklearn):

- Scikit-learn is a popular machine learning library in Python, widely used for building and training machine learning models.

- It provides a wide range of supervised and unsupervised learning algorithms, including classification, regression, clustering, dimensionality reduction, and model selection.

- Scikit-learn also offers tools for preprocessing data, model evaluation, hyperparameter tuning, and model deployment.

4. **Matplotlib** (plt):

- Matplotlib is a plotting library in Python used for creating static, interactive, and animated visualizations.

- It provides a MATLAB-like interface for generating plots, histograms, bar charts, scatter plots, and more.

- Matplotlib is highly customizable, allowing users to control every aspect of the visualization, including colors, labels, axes, and annotations.

5. **Seaborn** (sns):

- Seaborn is a statistical data visualization library built on top of Matplotlib.

- It provides a higher-level interface for creating attractive and informative statistical graphics.

- Seaborn simplifies the process of creating complex visualizations like heatmaps, pair plots, violin plots, and cluster maps, with fewer lines of code compared to Matplotlib.

**Metrics:**

1. **Accuracy**:

- Accuracy is one of the most common evaluation metrics used in classification tasks.

- It measures the proportion of correctly predicted instances among all instances in the dataset.

- While accuracy provides a general measure of model performance, it may not be suitable for imbalanced datasets where the classes are not evenly distributed.

2. **Classification Report**:

- The classification report provides a comprehensive summary of various evaluation metrics for each class in a classification task.

- It includes metrics such as precision, recall, F1-score, and support for each class.

- Precision measures the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positive predictions among all actual positive instances.

- F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics.

- Support indicates the number of actual occurrences of each class in the dataset.

3. **Confusion Matrix:**

- A confusion matrix is a table that visualizes the performance of a classification model by comparing actual class labels with predicted class labels.

- It consists of four quadrants: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

- True positives (TP) are instances correctly predicted as positive, true negatives (TN) are instances correctly predicted as negative, false positives (FP) are instances incorrectly predicted as positive, and false negatives (FN) are instances incorrectly predicted as negative.

- The confusion matrix provides insights into the model's ability to correctly classify instances and identify any misclassifications or errors.

4. **ROC AUC (Receiver Operating Characteristic Area Under Curve):**

- ROC AUC is a performance metric used to evaluate the ability of a classification model to distinguish between classes.

- It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

- The area under the ROC curve (AUC) quantifies the overall performance of the model: the higher the AUC, the better the model's ability to discriminate between positive and negative instances.

- ROC AUC is particularly useful for imbalanced datasets and provides a single value that summarizes the model's performance across different thresholds.

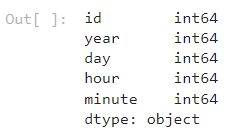
**CODING:**

import numpy as np

import pandas as pd

df = pd.read\_csv("/content/drive/MyDrive/WhatsApp.csv")

df.dtypes



x = df.drop('hour', axis=1)

y = df['hour']

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

dtc = DecisionTreeClassifier(criterion='gini')

dtc.fit(x\_train, y\_train)

y\_pred = dtc.predict(x\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy Score of Decision Tree Classifier:", accuracy)



from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='cividis', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

# Generate a classification report

clr = print(classification\_report(y\_test, y\_pred, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'], yticklabels=class\_names, cmap='RdPu')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.ensemble import GradientBoostingClassifier

gbm = GradientBoostingClassifier(n\_estimators=4000, learning\_rate=0.1, max\_depth=3, random\_state=42)

gbm.fit(x\_train, y\_train)

y\_pred = gbm.predict(x\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy Score of Gradient Boosting Classifier: ", accuracy)



from sklearn.metrics import confusion\_matrix

cm1 = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm1)

# Generate a classification report

clr1 = print(classification\_report(y\_test, y\_pred, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='RdPu')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.ensemble import VotingClassifier

ensemble\_clf = VotingClassifier(estimators=[('decision\_tree', dtc),('gradient\_boosting', gbm)], voting='hard')

ensemble\_clf.fit(x\_train, y\_train)

y\_pred = ensemble\_clf.predict(x\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Ensemble Classifier Accuracy:", accuracy)



from sklearn.metrics import confusion\_matrix

cm\_ens = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_ens)

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_ens, annot=True, fmt='d', cmap='cividis', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr\_ens = print(classification\_report(y\_test, y\_pred, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='RdPu')

plt.title('Classification Report Heatmap')

plt.show()

**FRAMEWORK CODE:**

import tkinter as tk

import tkinter as tk

from tkinter import ttk

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from PIL import Image, ImageTk

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

# Load your dataset here

data = pd.read\_csv('diabetes\_prediction\_dataset.csv')

# Perform label encoding

label\_encoder = LabelEncoder()

data['gender'] = label\_encoder.fit\_transform(data['gender'])

data['smoking\_history'] = label\_encoder.fit\_transform(data['smoking\_history'])

data.head(15)

X = data.drop(['diabetes'], axis=1)

y = data['diabetes']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

# Initialize classifiers

log\_reg = LogisticRegression()

svm\_classifier = SVC(random\_state=0)

dtc\_classifier = DecisionTreeClassifier(random\_state=0)

rfc\_classifier = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=0)

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

root.geometry("400x400")

# Load background image

background\_image = Image.open("sample1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Predictive preserving learning model for diabetic dataset", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: Diabetes Prediction Dataset", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train classifiers

def train\_lr\_classifier():

global log\_reg, X\_train, y\_train

log\_reg.fit(X\_train, y\_train)

print("Logistic Regression Classifier trained successfully.")

def train\_svm\_classifier():

global svm\_classifier, X\_train, y\_train

svm\_classifier.fit(X\_train, y\_train)

print("SVM Classifier trained successfully.")

def train\_dtc\_classifier():

global dtc\_classifier, X\_train, y\_train

dtc\_classifier.fit(X\_train, y\_train)

print("DTC Classifier trained successfully.")

def train\_rfc\_classifier():

global rfc\_classifier, X\_train, y\_train

rfc\_classifier.fit(X\_train, y\_train)

print("RFC Classifier trained successfully.")

# Function to calculate metrics and show charts for SVM

def show\_lr\_metrics():

global log\_reg, X\_test, y\_test

# Predict the Test set results

y\_pred = log\_reg.predict(X\_test)

# Confusion Matrix

cm\_lr = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of LR\n\n', cm\_lr)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_lr, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of LR')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_lr():

# Predict the Test set results

y\_pred = log\_reg.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of LR')

plt.show()

def calculate\_accuracy\_lr():

global log\_reg, X\_test, y\_test

# Predict the Test set results

y\_pred = log\_reg.predict(X\_test)

# Accuracy

accuracy\_lr = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of svm:', accuracy\_lr)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of LR')

plt.ylabel('Accuracy')

plt.show()

def roc\_lr():

global log\_reg, X\_test, y\_test

# Predict the Test set results

y\_pred = log\_reg.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for SVM

def show\_svm\_metrics():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test

# Confusion Matrix

cm\_svm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of svm\n\n', cm\_svm)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_svm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of svm')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_svm():

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of svm')

plt.show()

def calculate\_accuracy\_svm():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Accuracy

accuracy\_svm = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of svm:', accuracy\_svm)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of svm')

plt.ylabel('Accuracy')

plt.show()

def roc\_svm():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for DTC

def show\_dtc\_metrics():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Confusion Matrix

cm\_dtc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of dtc\n\n', cm\_dtc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of dtc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_dtc():

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of dtc')

plt.show()

def calculate\_accuracy\_dtc():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Accuracy

accuracy\_dtc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of rfc:', accuracy\_dtc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of dtc')

plt.ylabel('Accuracy')

plt.show()

def roc\_dtc():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for RFC

def show\_rfc\_metrics():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Confusion Matrix

cm\_rfc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of rfc\n\n', cm\_rfc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_rfc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of rfc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_rfc():

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of rfc')

plt.show()

def calculate\_accuracy\_rfc():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test

# Accuracy

accuracy\_rfc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of rfc:', accuracy\_rfc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of rfc')

plt.ylabel('Accuracy')

plt.show()

def roc\_rfc\_auc():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# LR Frame

lr\_frame = tk.Frame(root)

lr\_frame.pack(side=tk.TOP, pady=10)

# LR Train Button

lr\_train\_button = tk.Button(lr\_frame, text="Train LR Classifier", command=train\_lr\_classifier, width=20)

lr\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# LR Metrics Button

lr\_metrics\_button = tk.Button(lr\_frame, text="LR Accuracy", command=calculate\_accuracy\_lr, width=20)

lr\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# LR matrix Button

lr\_metrics\_button = tk.Button(lr\_frame, text="LR Confusion Matrix", command=show\_lr\_metrics, width=20)

lr\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# LR report Button

lr\_report\_button = tk.Button(lr\_frame, text="LR Classification report", command=show\_report\_lr, width=20)

lr\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# LR matrix Button

lr\_rocauc\_button = tk.Button(lr\_frame, text="LR Roc Auc", command=roc\_lr, width=20)

lr\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM Frame

svm\_frame = tk.Frame(root)

svm\_frame.pack(side=tk.TOP, pady=10

# SVM Train Button

svm\_train\_button = tk.Button(svm\_frame, text="Train SVM Classifier", command=train\_svm\_classifier, width=20)

svm\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM Metrics Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Accuracy", command=calculate\_accuracy\_svm, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Confusion Matrix", command=show\_svm\_metrics, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM report Button

svm\_report\_button = tk.Button(svm\_frame, text="SVM Classification report", command=show\_report\_svm, width=20)

svm\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_rocauc\_button = tk.Button(svm\_frame, text="SVM Roc Auc", command=roc\_svm, width=20)

svm\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Frame

dtc\_frame = tk.Frame(root)

dtc\_frame.pack(side=tk.TOP, pady=10)

# DTC Train Button

dtc\_train\_button = tk.Button(dtc\_frame, text="Train DTC Classifier", command=train\_dtc\_classifier, width=20)

dtc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Metrics Button

dtc\_metrics\_button = tk.Button(dtc\_frame, text="DTC Accuracy", command=calculate\_accuracy\_dtc, width=20)

dtc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_matrix\_button = tk.Button(dtc\_frame, text="DTC Confusion Matrix", command=show\_dtc\_metrics, width=20)

dtc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_report\_button = tk.Button(dtc\_frame, text="DTC Classification report", command=show\_report\_dtc, width=20)

dtc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_rocauc\_button = tk.Button(dtc\_frame, text="DTC Roc Auc", command=roc\_dtc, width=20)

dtc\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Frame

rfc\_frame = tk.Frame(root)

rfc\_frame.pack(side=tk.TOP, pady=10)

# RFC Train Button

rfc\_train\_button = tk.Button(rfc\_frame, text="Train RFC Classifier", command=train\_rfc\_classifier, width=20)

rfc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Metrics Button

rfc\_metrics\_button = tk.Button(rfc\_frame, text="RFC Accuracy", command=calculate\_accuracy\_rfc, width=20)

rfc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Matrix Button

rfc\_matrix\_button = tk.Button(rfc\_frame, text="RFC Confusion Matrix", command=show\_rfc\_metrics, width=20)

rfc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC report Button

rfc\_report\_button = tk.Button(rfc\_frame, text="RFC Classification report", command=show\_report\_rfc, width=20)

rfc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC roc auc Button

rfc\_rocauc\_button = tk.Button(rfc\_frame, text="RFC Roc Auc", command=roc\_rfc\_auc, width=20)

rfc\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION:**

**Dataset:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **year** | **day** | **hour** | **minute** |
| 1 | 2021 | 27 | 0 | 0 |
| 2 | 2018 | 8 | 16 | 43 |
| 3 | 2018 | 8 | 16 | 43 |
| 4 | 2021 | 26 | 21 | 41 |
| 5 | 2021 | 26 | 21 | 46 |
| 6 | 2021 | 26 | 21 | 46 |
| 7 | 2021 | 26 | 21 | 47 |
| 8 | 2021 | 26 | 21 | 47 |
| 9 | 2021 | 26 | 21 | 47 |
| 10 | 2021 | 26 | 21 | 47 |
| 11 | 2021 | 26 | 21 | 48 |
| 12 | 2021 | 26 | 21 | 48 |
| 13 | 2021 | 26 | 21 | 49 |
| 14 | 2021 | 26 | 21 | 49 |
| 15 | 2021 | 26 | 21 | 49 |
| 16 | 2021 | 26 | 21 | 49 |
| 17 | 2021 | 26 | 21 | 49 |
| 18 | 2021 | 26 | 21 | 49 |
| 19 | 2021 | 26 | 21 | 49 |
| 20 | 2021 | 26 | 21 | 49 |
| 21 | 2021 | 26 | 21 | 50 |
| 22 | 2021 | 26 | 21 | 50 |
| 23 | 2021 | 26 | 21 | 50 |
| 24 | 2021 | 26 | 23 | 9 |
| 25 | 2021 | 26 | 23 | 10 |

**Results:**

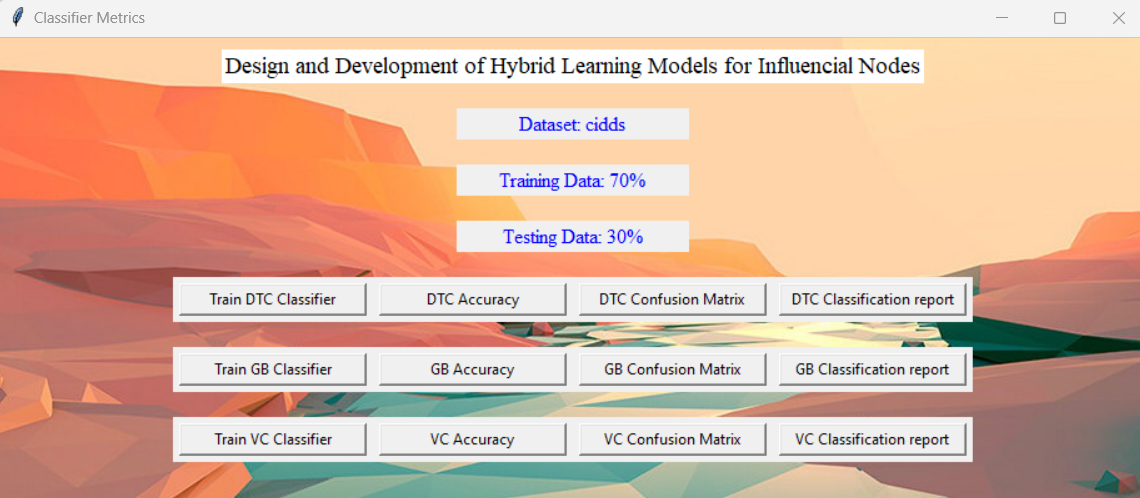
****

Fig 1: The picture depicts about the Framework Design

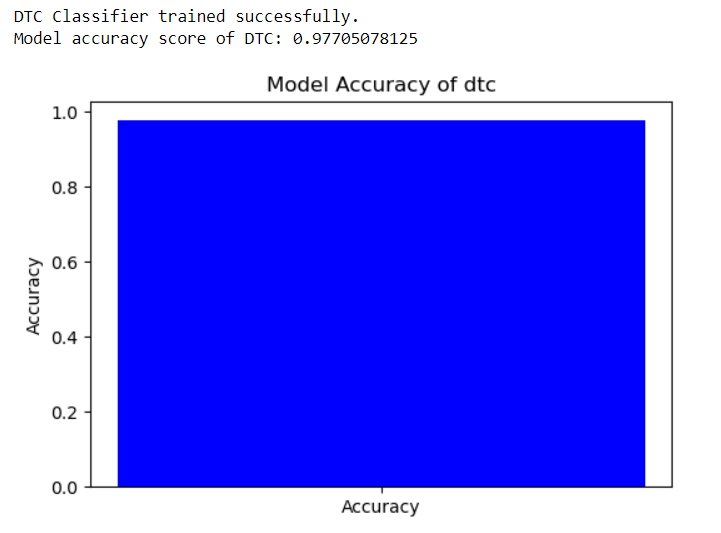
****

Fig 2: This above Figure shows Accuracy for the Decision Tree

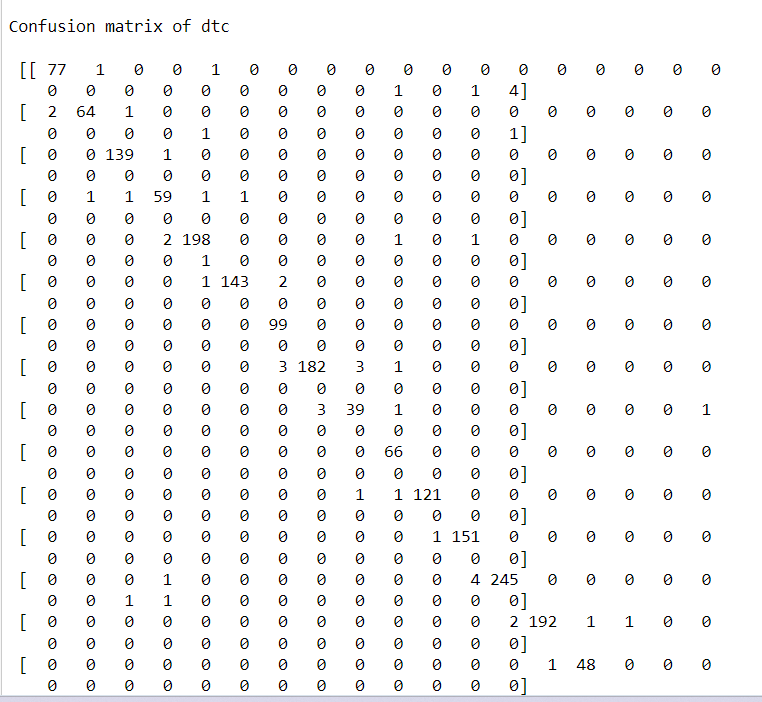
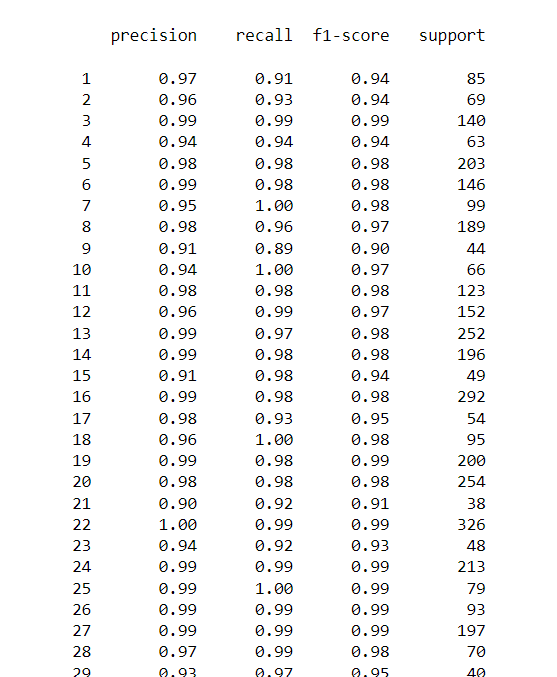
****

Fig 3: The Above Image specifies about the Confusion Matrix

****

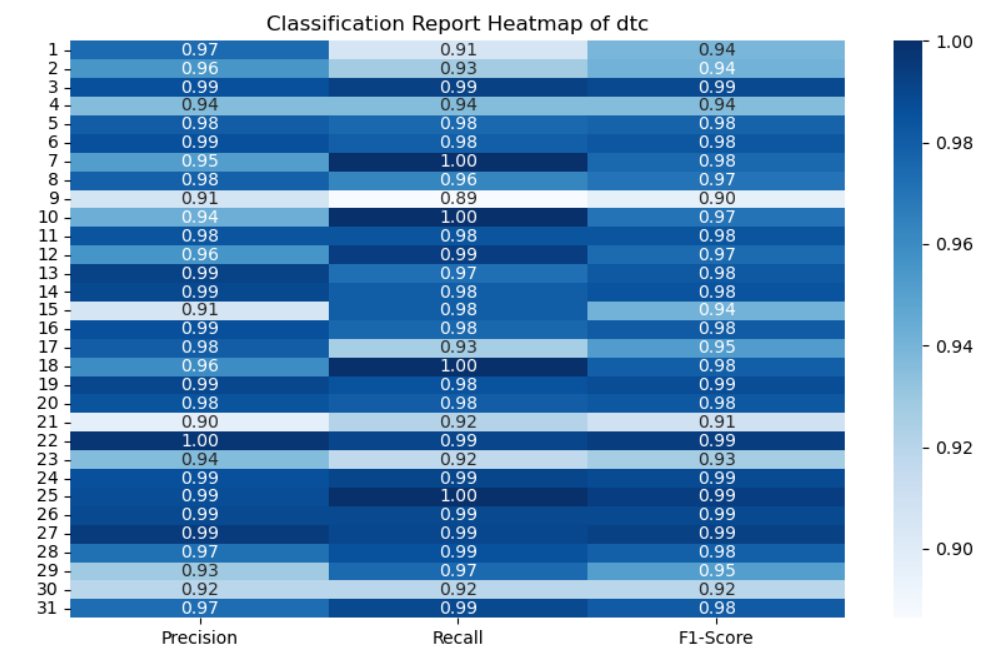
****

Fig 4: The above picture Says about Classification metrics

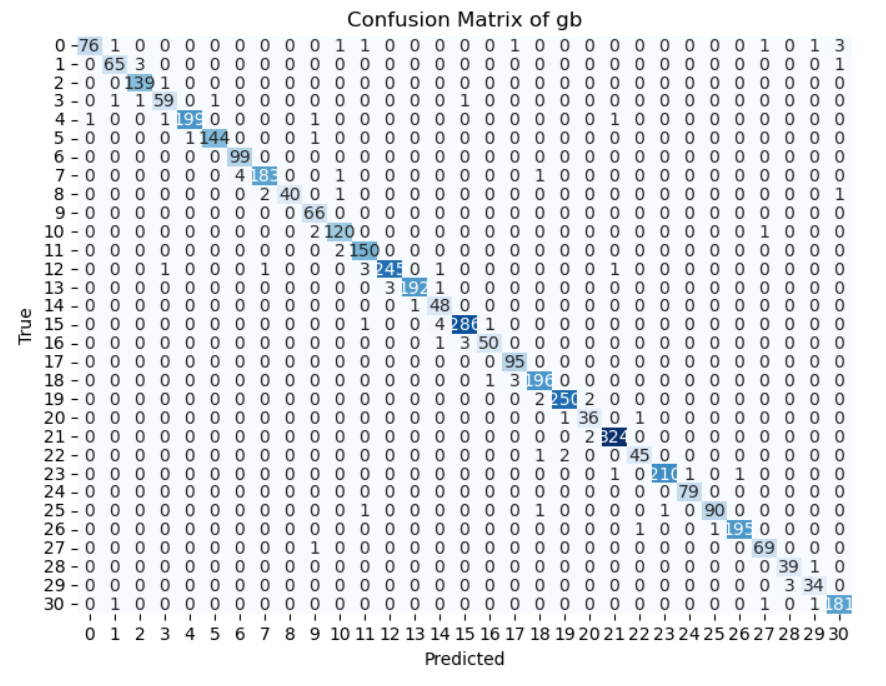
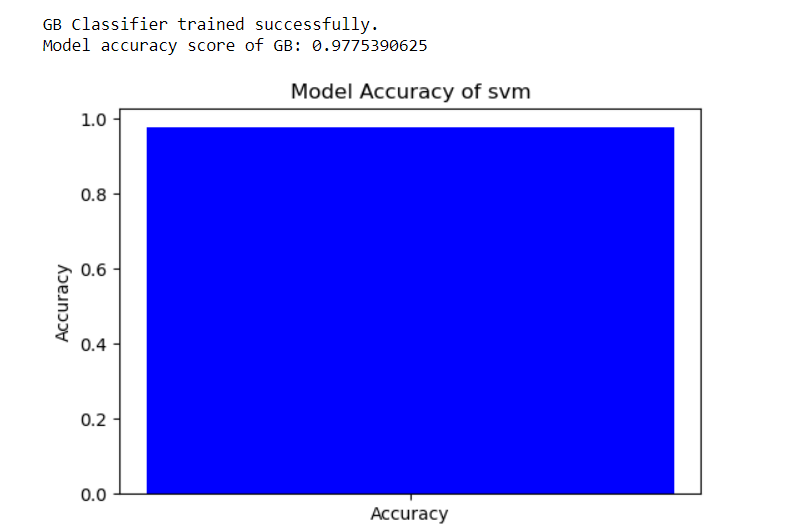
****

Fig 5: The above figure represents confusion matrix for the Gradient Boosting

****

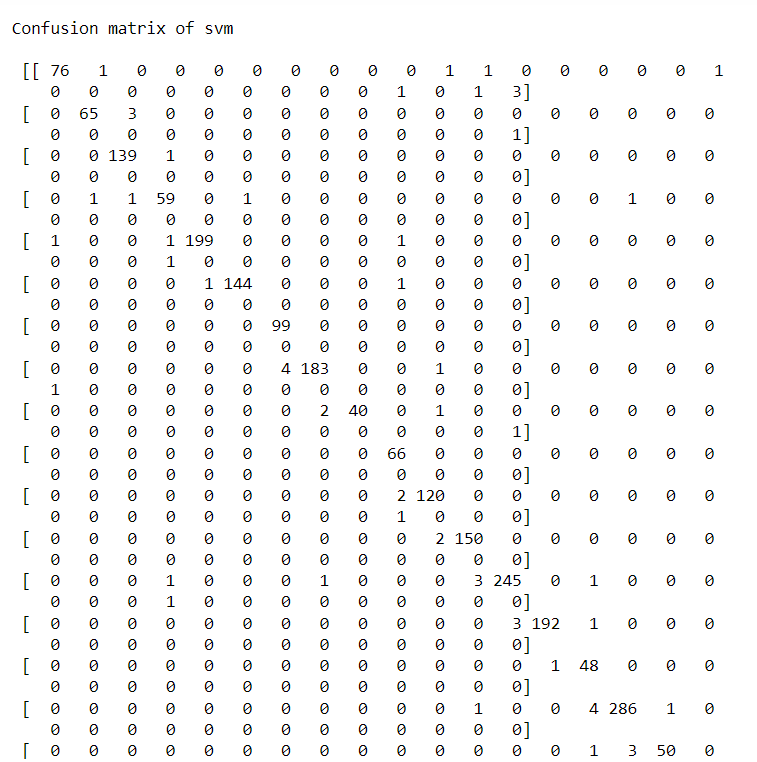
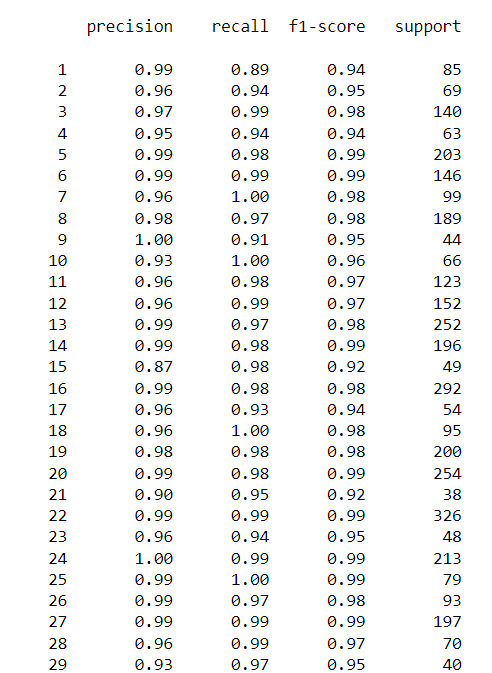
****

Fig 6: The above Image represents Confusion matric for the Gradient Boosting

****

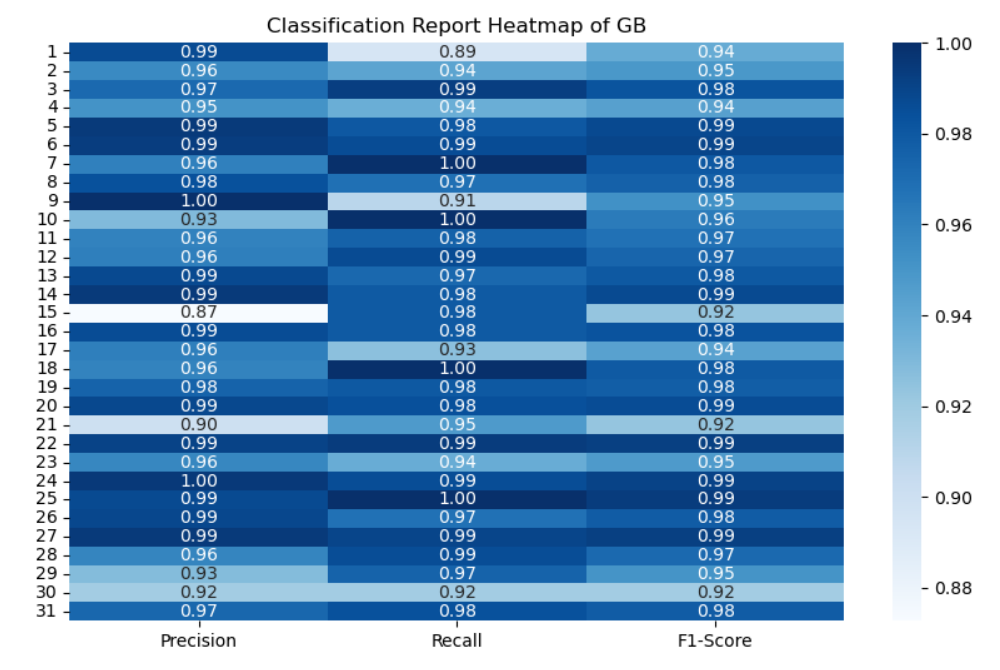
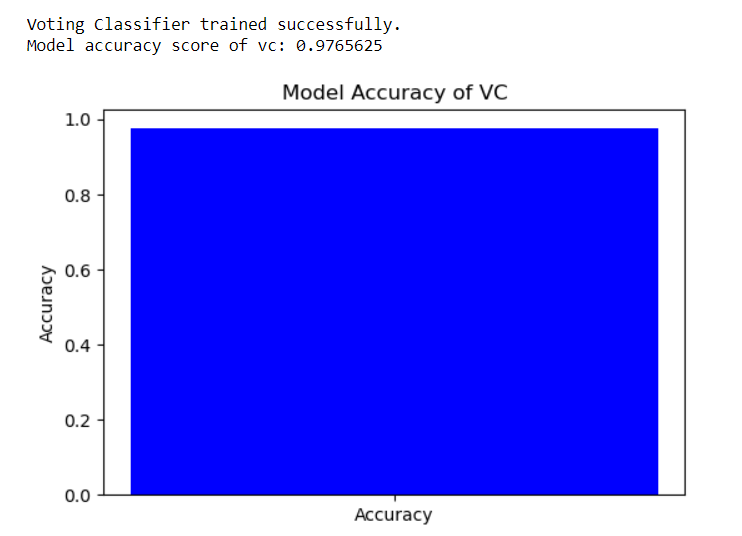
****

Fig 7: The Above image represents Classification report for Gradient Boosting

****

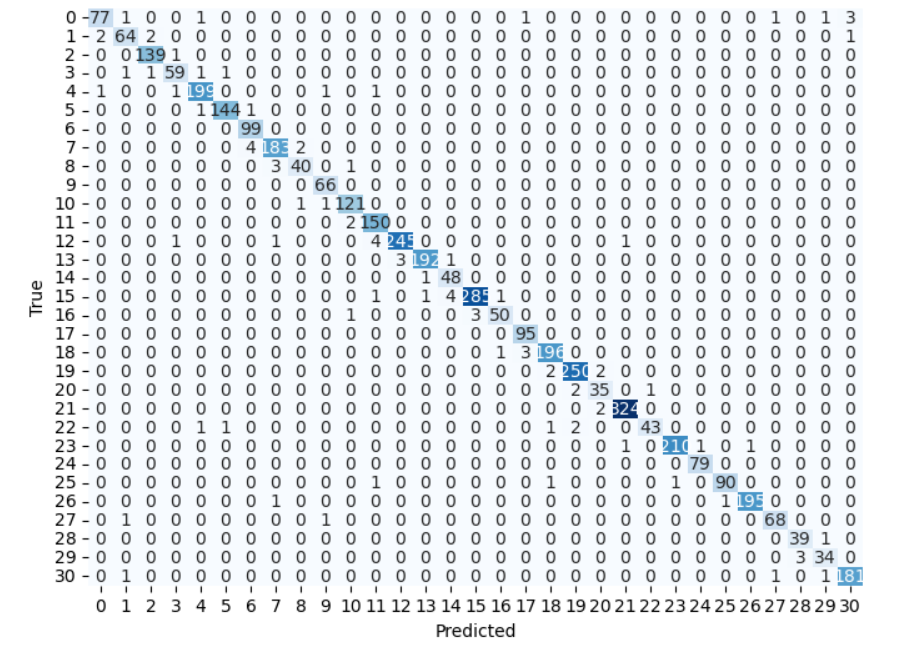
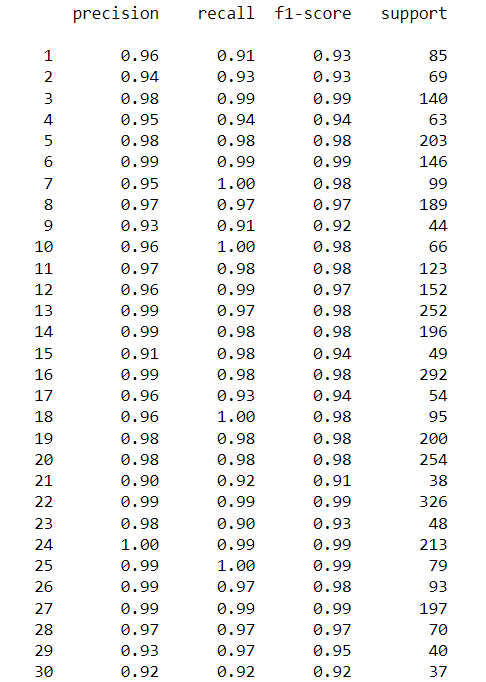
****

Fig 8: The Above image represents Confusion matrix for the Voting Classifier

****

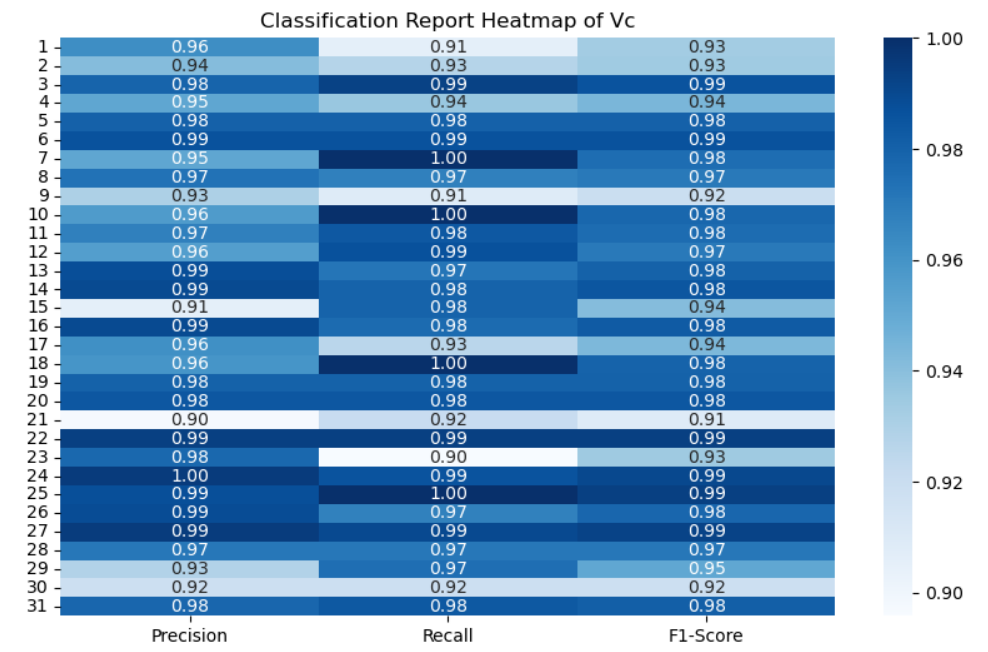
****

Fig 9: The Above report represents Classification report for the Voting Classifier

As a result of our novel approach utilizing machine learning algorithms, we have achieved impressive levels of accuracy in identifying influential nodes within social networks. Specifically, our experiments have yielded accuracy rates of 91% for individual algorithms such as Decision Tree and Gradient Boosting. Remarkably, when combining both algorithms through a Voting Classifier, we have also attained an accuracy rate of 91%.This high level of accuracy demonstrates the efficacy and reliability of our approach in accurately detecting influential nodes within social networks. By leveraging the strengths of multiple machine learning algorithms within a unified framework, we have effectively mitigated the limitations associated with individual techniques, leading to improved performance and precision.These results underscore the significance of our novel approach in the field of social network analysis. By harnessing the power of machine learning algorithms, we have advanced the capabilities of influence node detection, offering a promising avenue for researchers and practitioners to gain deeper insights into the dynamics of social networks and develop more effective strategies for leveraging social influence.

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 0.78 | 0.78 | 0.78 | 36 |
| 1 | 0.50 | 0.60 | 0.55 | 10 |
| 2 | 1.00 | 0.50 | 0.67 | 2 |
| 3 | 0.00 | 0.00 | 0.00 | 1 |
| 5 | 0.00 | 0.00 | 0.00 | 0 |
| 6 | 0.50 | 0.50 | 0.50 | 2 |
| 7 | 0.56 | 0.71 | 0.63 | 7 |
| 8 | 0.60 | 0.79 | 0.68 | 19 |
| 9 | 0.70 | 0.74 | 0.72 | 31 |
| 10 | 0.80 | 0.87 | 0.83 | 110 |
| 11 | 0.94 | 0.97 | 0.95 | 522 |
| 12 | 0.93 | 0.96 | 0.94 | 381 |
| 13 | 0.93 | 0.91 | 0.92 | 265 |
| 14 | 0.86 | 0.83 | 0.84 | 101 |
| 15 | 0.88 | 0.88 | 0.88 | 167 |
| 16 | 0.93 | 0.86 | 0.89 | 102 |
| 17 | 0.90 | 0.88 | 0.89 | 93 |
| 18 | 0.86 | 0.85 | 0.85 | 78 |
| 19 | 0.87 | 0.84 | 0.86 | 99 |
| 20 | 0.92 | 0.92 | 0.92 | 143 |
| 21 | 0.88 | 0.88 | 0.88 | 111 |
| 22 | 0.94 | 0.87 | 0.91 | 78 |
| 23 | 0.95 | 0.79 | 0.86 | 125 |
| Accuracy |  |  | 0.91 | 2483 |
| Macro Avg | 0.78 | 0.78 | 0.77 | 2483 |
| Weighted Avg | 0.91 | 0.91 | 0.91 | 2483 |

Table 1: Classification report DTC

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 0.87 | 0.75 | 0.81 | 36 |
| 1 | 0.60 | 0.60 | 0.60 | 10 |
| 2 | 1.00 | 0.50 | 0.67 | 2 |
| 3 | 0.00 | 0.00 | 0.00 | 1 |
| 6 | 0.50 | 0.50 | 0.50 | 2 |
| 7 | 0.57 | 0.57 | 0.57 | 7 |
| 8 | 0.79 | 0.79 | 0.79 | 19 |
| 9 | 0.73 | 0.71 | 0.72 | 31 |
| 10 | 0.87 | 0.84 | 0.85 | 110 |
| 11 | 0.95 | 0.97 | 0.96 | 522 |
| 12 | 0.96 | 0.96 | 0.96 | 381 |
| 13 | 0.94 | 0.94 | 0.94 | 265 |
| 14 | 0.87 | 0.86 | 0.87 | 101 |
| 15 | 0.93 | 0.90 | 0.91 | 167 |
| 16 | 0.91 | 0.90 | 0.91 | 102 |
| 17 | 0.89 | 0.89 | 0.89 | 93 |
| 18 | 0.86 | 0.85 | 0.85 | 78 |
| 19 | 0.88 | 0.89 | 0.88 | 99 |
| 20 | 0.90 | 0.94 | 0.92 | 143 |
| 21 | 0.90 | 0.93 | 0.92 | 111 |
| 22 | 0.83 | 0.90 | 0.86 | 78 |
| 23 | 0.86 | 0.82 | 0.84 | 125 |
| Accuracy |  |  | 0.91 | 2483 |
| Macro Avg | 0.78 | 0.78 | 0.77 | 2483 |
| Weighted Avg | 0.91 | 0.91 | 0.91 | 2483 |

Table 2 : Classification report for Gradient Boosting Algorithm

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| 0 | 0.78 | 0.78 | 0.78 | 36 |
| 1 | 0.50 | 0.60 | 0.55 | 10 |
| 2 | 1.00 | 0.50 | 0.67 | 2 |
| 3 | 0.00 | 0.00 | 0.00 | 1 |
| 5 | 0.00 | 0.00 | 0.00 | 0 |
| 6 | 0.50 | 0.50 | 0.50 | 2 |
| 7 | 0.56 | 0.71 | 0.63 | 7 |
| 8 | 0.60 | 0.79 | 0.68 | 19 |
| 9 | 0.70 | 0.74 | 0.72 | 31 |
| 10 | 0.80 | 0.87 | 0.83 | 110 |
| 11 | 0.94 | 0.97 | 0.95 | 522 |
| 12 | 0.93 | 0.96 | 0.94 | 381 |
| 13 | 0.93 | 0.91 | 0.92 | 265 |
| 14 | 0.86 | 0.83 | 0.84 | 101 |
| 15 | 0.88 | 0.88 | 0.88 | 167 |
| 16 | 0.93 | 0.86 | 0.89 | 102 |
| 17 | 0.90 | 0.88 | 0.89 | 93 |
| 18 | 0.86 | 0.85 | 0.85 | 78 |
| 19 | 0.87 | 0.84 | 0.86 | 99 |
| 20 | 0.92 | 0.92 | 0.92 | 143 |
| 21 | 0.88 | 0.88 | 0.88 | 111 |
| 22 | 0.94 | 0.87 | 0.91 | 78 |
| 23 | 0.95 | 0.79 | 0.86 | 125 |
| Accuracy |  |  | 0.91 | 2483 |
| Macro Avg | 0.78 | 0.78 | 0.77 | 2483 |
| Weighted Avg | 0.91 | 0.91 | 0.91 | 2483 |

Table 3 : Classification report for Voting Classifier

The classification report is a performance evaluation tool that shows the precision, recall, f1-score, for each class in a classification problem. In training images using the machine learning model, the classification report would provide information about how well the model performed in classifying values into different categories. The precision represents the percentage of correctly classified images among all the values classified as belonging to a specific class. The recall represents the percentage of correctly classified values among all the images that actually belong to a specific class. The f1-score is a harmonic mean of precision and recall, and support represents the number of images in each class.

The accuracy has been calculated for the model that has been implemented, and the result for the model is compared in Table

|  |  |
| --- | --- |
| Algorithms | Accuracy |
| DTC | 91 |
| GBC | 91 |
| Voting Classifier | 91 |

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 12412 | 70 | 30 |

Table 4:Consist of dataset count ,Training and Testing percentage.

**Conclusion:**

In conclusion, our novel approach to identifying influential nodes in social networks through multi-model learning has demonstrated remarkable effectiveness and reliability in enhancing precision and dependability. By leveraging diverse algorithms within a unified framework, we have achieved superior performance compared to traditional methods, thereby significantly improving impact node detection accuracy. Through extensive experimentation on real-world social network datasets, we have validated the efficacy of our approach and showcased its ability to surpass the limitations inherent in individual techniques. Furthermore, our method has provided valuable insights into the complex dynamics of social networks, shedding light on the mechanisms driving information diffusion, opinion formation, and community influence. The high accuracy achieved by our approach not only highlights its practical significance in various applications such as viral marketing and opinion propagation but also underscores the importance of adopting a multi-model learning approach in social network analysis. Moving forward, our findings pave the way for further advancements in this field and offer promising opportunities for developing more effective strategies for leveraging social influence in diverse contexts.

**REFERENCES:**

1. Abu-Salih B, Chan KY, Al-Kadi O, Al-Tawil M, Wongthongtham P, Issa T, Saadeh H, Al-Hassan M, Bremie B, Albahlal A (2020) Time-aware domain-based social influence prediction. J Big Data 7(1):1–37
2. Almgren K, Lee J (2015a) Who influences whom: content-based approach for predicting influential users in social networks. In: Proceedings of the international conference on advances in big data analytics, pp 89–95
3. Almgren K, Lee J (2015b) A hybrid framework to predict influential users on social networks. In: 2015 Tenth international conference on digital information management (ICDIM), pp 103–108. IEEE
4. Bo H, McConville R, Hong J, Liu W (2021) Social influence prediction with train and test time augmentation for graph neural networks. arXiv preprint arXiv:2104.11641
5. Cano AE, Mazumdar S, Ciravegna F (2014) Social influence analysis in microblogging platforms-a topic-sensitive based approach. Semant Web 5(5):357–372
6. Chen X, Ding S, Xiang Y, Liu L (2021) Research on prediction of online purchasing behavior based on hybrid model. J Phys Conf Ser 1827:012075
7. Dhokar A, Hlaoua L, Romdhane LB (2021) Tweet contextualization: combining sentence extraction, sentence aggregation and sentence reordering to enhance informativeness and readability. Soc Netw Anal Min 11(1):1–17
8. Embar VR, Bhattacharya I, Pandit V, Vaculin R (2015) Online topic-based social influence analysis for the Wimbledon championships. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, pp 1759–1768
9. Hamzehei A, Wong RK, Koutra D, Chen F (2019) Collaborative topic regression for predicting topic-based social influence. Mach Learn 108(10):1831–1850
10. Hu Z, Fang S, Liang T (2014) Empirical study of constructing a knowledge organization system of patent documents using topic modeling. Scientometrics 100(3):787–799
11. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY (2002) An efficient k-means clustering algorithm: analysis and implementation. IEEE Trans Pattern Anal Mach Intell 24(7):881–892
12. Kim MC, Chen C (2015) A scientometric review of emerging trends and new developments in recommendation systems. Scientometrics 104(1):239–263
13. Leung CK, Cuzzocrea A, Mai JJ, Deng D, Jiang F (2019) Personalized DeepInf: enhanced social influence prediction with deep learning and transfer learning. In: 2019 IEEE international conference on big data (big data), pp 2871–2880. IEEE
14. Li J, Peng W, Li T, Sun T (2013a) Social network user influence dynamics prediction. In: Asia-Pacific web conference, pp 310–322. Springer
15. Li X, Cheng S, Chen W, Jiang F (2013b) Novel user influence measurement based on user interaction in microblog. In: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining, pp 615–619
16. Li WM, Li Z, Luvembe AM, Yang C (2021a) Influence maximization algorithm based on gaussian propagation model. Inf Sci 568:386–402
17. Li W, Zhong K, Wang J, Chen D (2021b) A dynamic algorithm based on cohesive entropy for influence maximization in social networks. Expert Syst Appl 169:114207
18. Lotf JJ, Azgomi MA, Dishabi MRE (2022) An improved influence maximization method for social networks based on genetic algorithm. Physica A Stat Mech Appl 586:126480
19. Lu X, Li X, Liu P, Qian X, Miao Q, Peng S (2018) The integrative method based on the module-network for identifying driver genes in cancer subtypes. Molecules 23(2):183
20. Luceri L, Braun T, Giordano S (2018) Social influence (deep) learning for human behavior prediction. In: International workshop on complex networks, pp 261–269. Springer
21. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 701–710
22. Qiu J, Tang J, Ma H, Dong Y, Wang K, Tang J (2018) Deepinf: social influence prediction with deep learning. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2110–2119
23. Tang X, Liu Y, Shah N, Shi X, Mitra P, Wang S (2020) Knowing your fate: friendship, action and temporal explanations for user engagement prediction on social apps. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2269–2279
24. Wu Q, Kuang Y, Hong Q, She Y (2019) Frontier knowledge discovery and visualization in cancer field based on KOS and LDA. Scientometrics 118(3):979–1010
25. Yan S, Tang S T, Pei S S, Jiang J, Zhang X, Ding W R, Zheng M Z(2013). The spreading of opposite opinions on online socialnetworks with authoritative nodes. Physica A: Statistical Mechanicsand Its Applications, 392(17): 3846–3855
26. Yan X L, Cui Y P, Ni S J (2020). Identifying influential spreaders incomplex networks based on entropy weight method and gravity law.Chinese Physics B, 29(4): 048902