PROJECT REPORT

Predicting Crime and Proposing Safer Neighborhoods: Using Machine Learning Models

Name: Vinay Devabhaktuni (801409486)

Primary Paper:

Author(s): Walczak, Steven

Title: Predicting Crime and Other Uses of Neural Networks in Police Decision Making

Year: 2021

Conference/Journal: Journal of Artificial Intelligence and Law (Springer)

Link: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8529125/

Abstract

Neighborhood safety and security have become top concerns for both residents and government in a time of growing urbanization and changing socioeconomic realities. Traditional approaches frequently rely on reactive actions, which might not be sufficient to meet newly emerging threatsor shifting dynamics. The aim of this project is to proactively identify regions with a higher risk of criminal behavior and to propose safer neighborhoods by creating predictive models. By usingmachine learning and data-driven techniques we will analyze historical crime data to make informed predictions to identify crime focused regions while also enabling the execution of focused interventions before incidents occur. We aim to develop and compare various classification models like Decision Tree, Random Forest, Artificial Neural Network (ANN), XGBoost algorithm. These models were selected for their proficiency in capturing diverse data patterns and enabling precise crime forecasts. Among the four machine learning models, random forest has a performance F1 score of 94 percent, XGBoost with 89 percent, and ANN at 91 percent. Decision Tree has outperformed with the highest F1 score of 94 percent. The outcome would be to increase public awareness about potentially high-risk areas and aid authorities in forecasting criminal activities at specific locations and times and help in effective utilization of police resources.

Key words: Crime Prediction, Criminal behavior forecasting, Focused Intervention, Neighborhood Safety, Aid Authorities

1. Introduction

1.1. Project Background and Execute Summary

Project Background, needs and importance

Crime is a multifaceted and intricate global phenomenon that has far-reaching effects on various communities worldwide. Its impact extends beyond the individuals directly affected, affecting their families and society as a whole. In addition to causing personal distress, crime leads to economic setbacks, social disturbances, and an overall deterioration in societal welfare. Countries are currently grappling with a variety of criminal activities, encompassing acts of violence such as homicides and assaults, as well as property crimes including burglaries and thefts. Recent statistical data indicates that crime rates have exhibited fluctuations over time; however, certain urban regions persistently confront considerable obstacles in ensuring public safety.

The most recent figures released by the FBI provide alarming statistics, indicating an approximate count of 1.1 million violent crimes and 7.5 million property crimes reported exclusively within the United States in the year 2024, thereby underscoring the magnitude of this problem Various factors contribute to the emergence and persistence of these issues, spanning socioeconomic inequalities, disparities in educational and employment opportunities, concerns related to substance abuse, as well as the ready availability of firearms. In particular, densely populated urban areas tend to witness higher incidences of criminal activities with varying degrees of risk amongst different communities.

The prevalence of poverty and unemployment significantly influences engagement in illegal actions as individuals facing economic hardships may resort to unlawful methods to meet their fundamental needs. Furthermore, societal inequities and limited access to high quality

education and healthcare can further compound crime-related challenges within marginalized neighborhoods. Drug-related offenses constitute a noteworthy facet of the crime problem in the United States. Criminal activities stemming from substance abuse and illicit drug trade encompass a spectrum that spans lesser charges such as petty drug possession to more severe crimes associated with organized criminal networks. Tackling drug-related crime usually requires an integrated strategy that encompasses law enforcement measures, rehabilitation interventions, and community-based initiatives.

The problem of gun violence is a matter of great significance. The accessibility of firearms, along with elements such as gang activity and mental health issues, contributes to a significant occurrence of crimes involving guns. Addressing this particular aspect of crime calls for an intricate strategy that involves not only implementing appropriate laws but also incorporating community-centered interventions aimed at minimizing gun-related violence. The prediction of criminal hotspots and crime type plays a crucial role in addressing safety concerns proactively in modern urban environments. Traditional reactive approaches are inadequate for tackling evolving threats. By enhancing public safety and optimizing resource allocation for law enforcement, this approach not only mitigates the economic, social, and psychological impacts of criminal activities but also leads to an improved quality of life for communities.

Targeted project problem and motivations

The project targets the limitations of conventional crime prevention measures in effectively reducing criminal activities in dynamic urban environments. Traditional methods, which rely heavily on law enforcement presence and reactive punitive actions, are often unable to adapt quickly enough to emerging threats and evolving patterns of unlawful behavior. This initiative seeks to implement an innovative approach that utilizes data analysis for early

detection of high-risk areas and times, allowing targeted interventions before incidents happen. Through these efforts, we aspire to revolutionize current crime prevention strategies, leading to safer communities and optimized allocation of law enforcement resources. Moreover, the advantages of the model extend to multiple sectors. Law enforcement agencies can effectively allocate their officers, city administrations can invest in crime prevention measures more efficiently, businesses can enhance their security protocols, and individuals can evaluate the risks present in their neighborhoods and take appropriate actions for personal and property safety. This initiative offers a groundbreaking approach to community safety by leveraging cutting-edge technology and adopting data-driven methodologies.

The project is motivated by the increasing influence of crime on communities worldwide, which calls for a fundamental change in strategies for preventing crime. Traditional approaches have revealed their limitations in effectively addressing emerging threats. The abundance of data and advancements in machine learning create an opportunity to predict criminal behavior proactively. By harnessing these technologies, our objective is to transform crime prevention and equip communities with the ability to protect their safety through anticipation. This initiative stems from our dedication to fostering secure neighborhoods, minimizing the societal impact of crime, and enabling law enforcement agencies with focused interventions based on data analysis.

Goals of the project

- **Crime Prevention Through Proactive Measures:** Create predictive models that can effectively detect areas and times with a higher likelihood of experiencing incidents, enabling targeted interventions in advance.
- **Resource Allocation Optimization:** Equip law enforcement agencies with databased

insights for optimizing resource allocation, thereby maximizing their effectiveness in reducing crime rates.

- **Public Safety and Awareness:** Enhance public consciousness regarding areas that may pose a heightened risk, empowering individuals to adopt precautionary actions for their personal safety.
- **Reduced Social and Economic Impact:** Alleviate the financial setbacks, community unrest, and emotional distress commonly associated with criminal activities through timely intervention and preventive measures.
- **Making Communities Safer:** Contribute towards the overall welfare and livability of urban settings by fostering feelings of security and confidence among residents.

Planned project approaches and method

1. **Data Collection and Preparation:** The dataset utilized in our project consists of historical crime data sourced from the official website of the United States of Government (Data.gov). A crucial step in the process involves thoroughly cleaning and preparing this dataset to ensure its consistency and accuracy. This may entail eliminating duplicate records, rectifying errors, as well as transforming the data into a standardized format.
2. **Feature Engineering:** After preparing the data, it is essential to construct features that are pertinent to the task of predicting criminal hotspots and type of crime. These features can be derived from several factors, including the nature of the crime, its timing and location, as well as demographic characteristics of the area. For instance, within our dataset for crime prediction purposes various engineered features could be considered such as:

- **Type of crime:** This feature can be used to pinpoint locations and times when specific sorts of crimes are more likely to occur.
- **Time of crime:** This feature can be used to identify patterns in criminal activity. For example, you might find that burglaries are more likely to occur during the day when people are at work or school.
- **Location of crime:** This feature can be used to identify specific areas where crime is more likely to occur. We could also use location data to identify factors that might contribute to crime, such as poverty or lack of opportunity.
- **Demographics of the area:** This feature can be used to identify groups of people who may be more vulnerable to crime. For example, we might find that certain crimes are more likely to occur in low-income neighborhoods or areas with a high concentration of young people.
- **Crime history:** This feature can be used to identify areas and times where crime has occurred in the past.

3. **Split the data into training and test sets:** In our approach, the dataset will be divided into two segments: a training set comprising 70% of the data and a test set accounting for the remaining 20%, with validation set being 10%. During the training phase, we input this training data into our chosen models. Through this process, the models acquire knowledge about complex patterns and underlying trends in the dataset. This crucial step enables accurate predictions based on features and historical crime data. Afterwards, we assess the performance of our model using the test set to ensure its ability to apply learned information effectively to new and

unseen data.

4. **Model selection and training:** After splitting the data, the models we're utilizing may be trained on the training set. This entails putting the data into the model and allowing it to understand the data's patterns and trends. We aim to develop and compare various classification models for crime prediction using this dataset. The models that we will consider are, Lin et al. (2017) has used Support Vector Machines which are effective for both classification and regression tasks, making them suitable for accurately predicting crimes such as burglary, robbery, and assault. These models identify a hyperplane in the feature space to differentiate between different classes and can effectively classify new data. V. Balu et al. (2022) uses K-nearest neighbors and provides a straightforward yet powerful method of classification. They determine the K most similar instances to a given sample and base their classification on the majority class among those neighbors. This approach is proven to be effective in crime prediction by finding close matches to new incidents. Logistic Regression estimates the probability of binary outcomes, such as occurrences of crimes, by utilizing historical data and influential factors. In addition, Bharati et al. (2018) used Random Forest algorithms improve crime prediction accuracy using multiple decision trees which contribute towards more reliable forecasts.
5. **Model evaluation:** After the training process, it is necessary to assess the model's performance on a separate set of data that was not used during the training phase. This evaluation serves as an estimation for how effectively the model will perform when applied to new and unseen data. Various metrics can be employed in

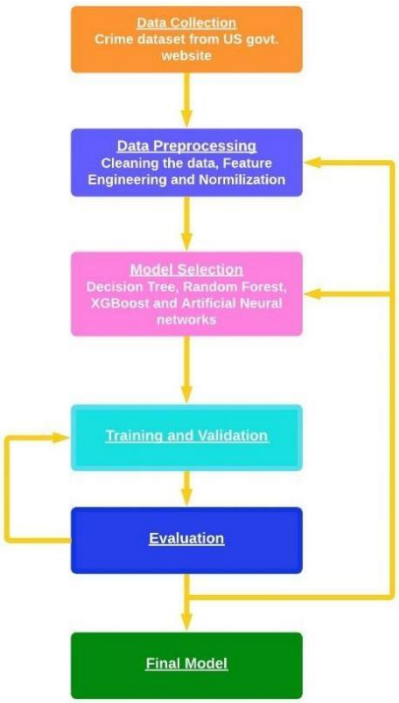
evaluating crime prediction models, with some commonly utilized ones being:

- Accuracy: The percentage of crimes that are correctly predicted by the model
- Precision: The percentage of predicted crimes that are actually crimes
- Recall: The percentage of actual crimes that are predicted by the model

6. **Model deployment:** After the evaluation process confirms that the model is functioning effectively, it can be implemented in a real-world setting. This implementation may include integrating the model with an established crime mapping software used by law enforcement agencies or creating a novel application to provide access to the model's predictions for both law enforcement and public use. Figure 1 shows complete model workflow consisting of all steps listed above.

Figure 1

Model Workflow



Note. Workflow diagram illustrating the journey from data collection to final model

Expected project contributions and application

This model will be beneficial for predicting crime and have the potential to make major contributions to public safety and crime prevention. Through the creation and implementation of precise and dependable crime prediction models, law enforcement agencies can enhance their resource deployment strategies, effectively deterring crimes before they occur. Consequently, these efforts have the potential to diminish overall crime rates, fostering secure communities and promoting social justice within society. The model extends to multiple applications, Police forces can strategically place their officers, distribute their resources, and create crime prevention plans using crime prediction models. Governments of cities can utilize crime prediction models to focus efforts in crime prevention, create initiatives for urban planning, and distribute funds for social services. Businesses can utilize crime prediction models to improve security and protect their employees and assets. Individuals can use models to estimate neighborhood threats and take the appropriate actions for personal and property safety.

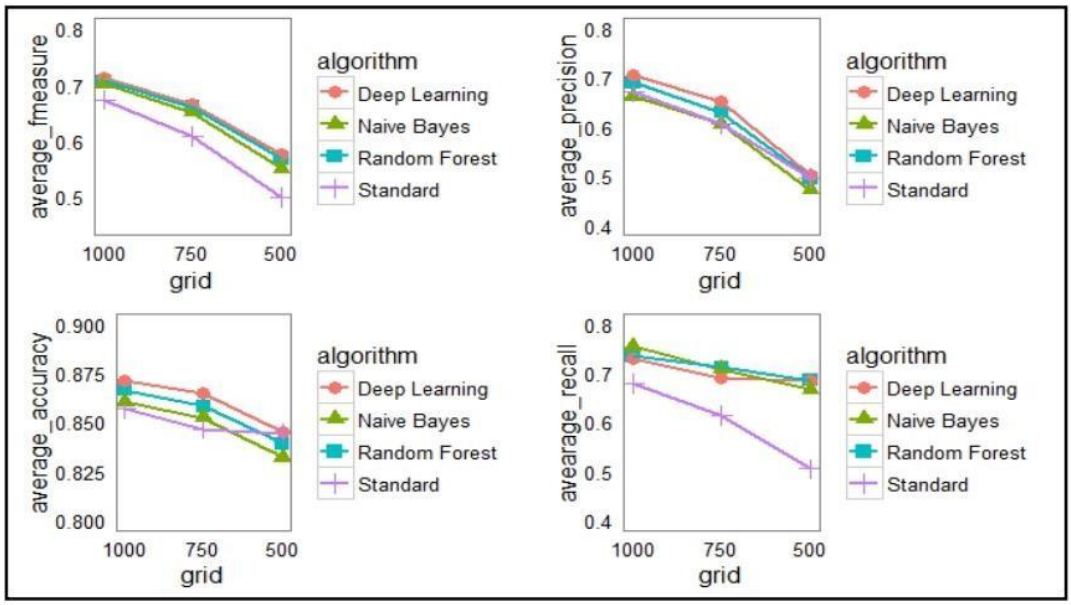
1.2. Technology and Solution Survey

After doing thorough surveys of current technologies, we found various solutions that matched closely with our problem statement. Lin et al. (2017) presents a data-driven approach to combat the increasing drug-related criminal activity in Taiwan using machine learning and spatial analysis. There is a theory called broken windows which means if we ignore any kind of low-level criminal activities some or other day it can lead to more serious crimes. In this paper, the team split the map into grids of various sizes, which had 56 features related to many different crime types and spatial-temporal patterns where spatial refers to space and temporal refers to time. They tried and tested many machine learning algorithms such as Deep Learning, Random Forest, and Naïve Bayes, with Deep Learning outperforming the others in terms of everything which includes accuracy, precision, recall, and f-measure. Technology tools used included R programming language and the H2O package emphasizing practical applicability for law enforcement agencies.

After summarizing and classifying the features and applications, we can observe from the figure below, the accuracy of all models (Deep Learning, Naive Bayes, Random Forest) is higher for larger grid sizes with highest accuracy of around 87% for deep learning models with grid size 1000x1000 as mentioned in Figure 2.

Figure 2

Average performance of different grid areas



Note. Different grid areas and their average performance.

Walczak et al. (2021) goes in detail about the application of neural networks. Doing survey on their paper, we can clearly notice that they tried to excel in solving classification and forecasting problems in sectors like law enforcement and crime prediction. The study used a huge substantial dataset consisting of 272,623 records sourced from the City of Detroit, Michigan, which included 38 distinct crime categories over 30 distinct zip codes. The paper suggests that NNs work well in valuable tools in big data environments. The complete goal of this paper was to collect a good amount of information, develop and evaluate neural networks for location and time-based prediction of crime types and then forecast crime locations. The paper has given more focus to attaining the issue of predictive accuracy. It reported a 16.4% accuracy rate in predicting 27 different crime types, and when crimes are grouped into seven categories, the accuracy increases to 27.1%. This accuracy takes out the random guessing factor and focuses more on statistical significance, reinforcing the practical utility of neural networks. These NN models are used to enhance law enforcement decision-making by giving real-time insights into the nature of crimes and potential locations which therefore enhances police response.

Divya et al. (2022) had a very different and innovative approach to increase security measures. They did this using deep learning. For feature extraction they used models like VGG-19 for object detection they used Faster R-CNN which resulted in accuracy more than 80%. Thus, we can say that this paper mainly focuses on extracting high-level features and then making bounded boxes for accurate object location. This paper covers a lot of things like crime detection, public safety in busy areas like banks, food malls, shopping centers, universities, etc., law enforcement support and customizable object detection. As this application has real-time data processing capabilities it has a faster response to security threats.

The study on crime prediction and analysis using machine learning was conducted by V. Balu et al. (2022) with the goal of predicting the likelihood of various crime types based on couple of factors like geographical and temporal. The paper mentions the application of machine learning models such as K-Nearest Neighbors (KNN), Decision Trees, and Random Forests. The selected features include location attributes (latitude and longitude), temporal elements (hour, day, month, year), and categorized crime types (e.g., Robbery, Gambling, Accident). The paper focuses on crime prediction offering a helping hand to law enforcement and public safety by facilitating proactive measures to prevent crimes and allocate resources more effectively.

After completing the technology and solution survey, I decided to employ Python packages for essential tasks such as EDA (exploratory data analysis), data cleaning, and data pre-processing. The data includes information about neighborhoods and crime incidents, so it's crucial to ensure it's clean and well-structured.

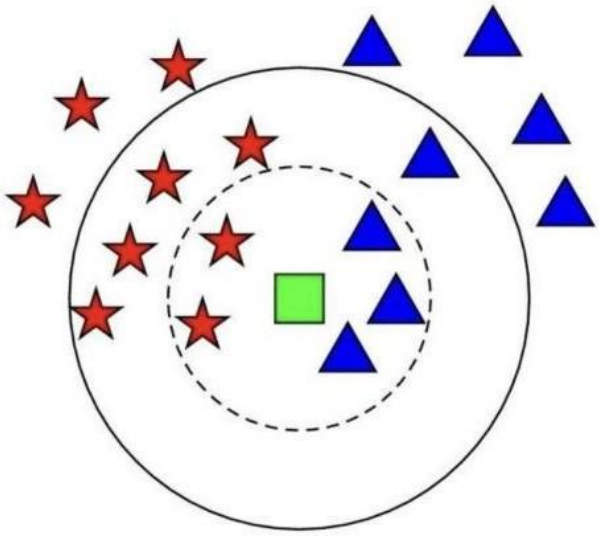
Additionally, based on the research papers mentioned in this section, we can observe that using classification and predictive models like Support Vector Machines (SVM), K-Nearest

Neighbors (K-NN), Logistic Regression and Random Forest algorithm will serve as a good technology solution for our use case. The most critical thing in developing a highly accurate model would be the concept of hyper parameter tuning the model which would result in higher levels of accuracy.

Further comparing various approaches, algorithms, and models it was observed that each algorithm has its strengths and weaknesses. SVM, for example, works well when dealing with smaller datasets and aims for higher accuracy, which could help us classify crime incidents effectively. KNN, on the other hand, is useful for pattern recognition in larger datasets and can help identify neighborhoods with similar characteristics as mentioned in Figure 3.

Figure 3

KNN principle diagram



Note. K-Nearest Neighbor Diagram.

1.3. Literature Survey of Existing Research

In the research paper titled 'Crime Prediction Using K-Nearest Neighbors Algorithm' Kumar et al. (2020) says the K-nearest neighbor algorithm is employed to predict the most

frequently occurring crime in a specific location. The dataset used in this study is a modified version of the original dataset, which was obtained by scraping data from the police website of a city in central India. To preprocess the data, the Extra Trees Classifier is utilized to assess the importance of features, and the least important features are subsequently discarded. Ultimately, only four features: date, time, latitude, and longitude are retained for predicting the most likely type of crime. The elbow method is employed to identify the optimal value of 'k', and the results indicate that k values within the range of 1 to 15 provides the best model performance. Notably, the model proposed by the author surpasses previous work, and it achieves the highest accuracy of 99% when 'k' is set to 3.

Ahishakiye et al. (2017) conducted a comprehensive comparison of various machine learning algorithms, including Naive Bayes, SVM, Decision Tree and Multi-layered Perceptron. After a thorough evaluation, they decided to use a decision tree algorithm to predict crime categories ('Low,' 'Medium,' and 'High') based on Number of Violent Crimes Per Population. To create this J48 classifier, they employed the Waikato Environment for Knowledge Analysis (WEKA) Toolkit and trained it using the 'Crime and Communities' dataset sourced from the UCI dataset repository. In the final model development, they deliberately narrowed down the attributes from the initial 128 to just 12, the selection process guided by their domain knowledge. The decision tree model they developed achieved an outstanding accuracy rate of 94.3%.

In the article titled 'Predicting Crime and Other Uses of Neural Networks in Police Decision Making,' Walczak et al. (2021) the authors highlight the capabilities of Neural Network (NN) models in analyzing crimes, which can potentially enhance the preparedness and response of police officers. The study employs two different NN training methods:

backpropagation and radial basis function (RBF), to predict either the location of a crime based on crime type and timestamp or predict the crime type given location and time information. In the crime type prediction process, clustering technology is employed to address the severe imbalance between different crime types. The accuracy increases from 16.4% to 27.1% when the number of crime type clusters is reduced from 27 to 7. It shows all the NN models exhibit superior performance in predicting the location of a crime as opposed to forecasting the type of crime. Among all the models, the single-hidden layer MLP achieved the highest accuracy of 31.2% in location prediction. The authors suggest that further investigation is necessary to achieve even higher performance in this domain.

In the research paper titled “Crime Prediction and Analysis Using Machine Learning,” Bharati et al. (2018) aimed for a crime prediction and analysis using machine learning which helps in crime prevention and law enforcement. They have used potential techniques such as clustering, data preprocessing, and location-based modeling for predicting and analyzing criminal activities effectively. For this research they have used Chicago crime dataset which is available in Kaggle and applied Machine learning algorithms such as KNN classification, Logistic Regression, Decision trees and Random Forest for crime prediction. Overall, the paper provides a foundation for understanding how machine learning can significantly contribute in the detection and prediction of crimes

In the paper “Crime forecasting: a machine learning and computer vision approach to crime prediction and prevention,” Shah et al. (2021) said that the integration of Machine Learning and computer vision for crime prediction can be greatly helpful. The authors have discussed such technologies are been successfully used and motivated for further research in this area. The paper describes the potential benefits of a security system that monitors the area

continuously and helps in preventing the crime more effectively.

The detection of crime intentions poses a significant challenge, yet it holds immense promise as a transformative tool in preventing criminal activities. Machine learning algorithms like VGGNet19 demonstrate high accuracy in identifying guns and knives held by individuals. Incorporating crime intention detection systems into closed circuit television cameras enables real-time identification of potential crimes. Nonetheless, the development of such systems encounters obstacles surrounding the availability of comprehensive and dependable datasets for model training and evaluation. Additionally, biases within models present further challenges that may result in inaccurate or unjust predictions. The CIDS exemplifies a noteworthy Crime Intention Detection System put forward by Divya et al. (2022) research paper which effectively utilizes the pre-trained VGGNET19 model to identify weapons being carried by individuals. The Crime Identification and Detection System demonstrated a notable accuracy rate of 79%, surpassing other approaches currently used for crime detection.

In the scholarly publication titled "Crime Prediction Utilizing Machine Learning and Deep Learning," authored by Krishnaveni K.P. and Dr. Manoharan S.S., an in-depth examination is undertaken to explore the potential of machine learning and deep learning techniques for predicting crime. The authors highlight how these technologies possess a transformative ability to identify intricate patterns and trends within crime data, alleviating the challenges associated with manual analysis alone. This newfound understanding facilitates real-time crime anticipation, empowering law enforcement agencies in maximizing their resource allocation accuracy. Numerous methodologies are extensively investigated throughout this research, including Support Vector Machines, Random Forests, Neural Networks, as well as sophisticated models such as Convolutional Neural Networks and Recurrent Neural Networks.

In addition, the authors also discuss the importance of accessing various types of important data for accurate crime prediction. This includes historical crime data, demographic information, socioeconomic indicators, and environmental factors. They acknowledge the potential impact that machine learning and deep learning can have in revolutionizing crime prediction but emphasize several challenges that need to be overcome. These challenges include obtaining large and reliable datasets, addressing biases in machine learning models, and ensuring interpretability of these models. The authors call for future research efforts to focus on developing robust machine learning and deep learning models that can effectively navigate these obstacles.

The paper titled "Machine Learning Techniques for Crime Prediction: A Review" by Dr. B.S. Jadhav et al. (2021) provides a comprehensive analysis of recent advancements in utilizing machine learning methods for forecasting criminal activities. The authors thoroughly explore various machine learning algorithms including Support Vector Machines, Random Forests, Decision Trees, Neural Networks, and Logistic Regression within this domain. Additionally, they examine the datasets used to evaluate these algorithms, citing well-known sources such as crime data from Chicago Police Department, Los Angeles Police Department, New York Police Department as well as the National Crime Victimization Survey. Moreover, significant attention is given to critical challenges associated with developing and deploying machine learning models for crime prediction which include limited availability of large-scale reliable datasets, potential biases embedded in the models themselves and complexities involved in interpreting their outputs. The authors assert that machine learning has great potential in crime prediction, but caution that there are challenges to address before it can be widely implemented. Additionally, the paper recognizes other methods like Natural Language Processing, Computer

Vision, and Geospatial Analysis as promising approaches for enhancing crime prediction through analyzing text, images, and geographical data. These models have various applications such as identifying areas with high levels of criminal activity, forecasting future crimes incidents detecting suspicious behavior and recognizing possible offenders. As a result they benefit stakeholders including law enforcement agencies security firms insurance companies and researchers. However, it is crucial to approach the application of machine learning in crime prediction with caution. Though this technology offers substantial advancements, there is a need for vigilance due to potential errors and biases. It is essential to follow responsible and ethical practices when using such tools.

A recent advancement in crime intention detection is presented in the paper by J. Arunnehru et al. (2021), titled "Crime Intention Detection Using Deep Learning." The author introduces a system that combines convolutional neural networks for feature extraction from video footage and support vector machines for classification. Through extensive evaluation on a dataset of 10,000 video clips, the system achieves a commendable accuracy rate of 95% in detecting criminal behavior. One of the notable advantages of this proposed system is its real-time detection capability, allowing for immediate action in response to potential criminal activity. Furthermore, the system demonstrates adapt. It is important to acknowledge that there are several aspects that require further investigation in relation to the system's performance on a bigger dataset, examination of potential biases, and evaluation of computational expenses for real-world implementation.

The paper "Detecting Criminal Intent in Early Stages of Shoplifting Cases Using 3D Convolutional Neural Networks" by Grega et al. (2013) presents a specialized 3D CNN model for the early detection of criminal intentions in shoplifting incidents. Trained on video footage

gathered from a retail environment, this novel approach effectively learns to identify suspicious behavior indicative of potential shoplifting occurrences. Remarkably, the 3D CNN achieves an impressive accuracy rate of 75% in detecting criminal intentions at the initial stages of shoplifting cases, surpassing previous methods that relied on manually engineered features and achieved accuracies around 60%. The authors emphasize several advantages offered by their proposed 3D CNN over existing systems for shoplifting detection, notably its ability to autonomously discern suspicious behavior without relying on predefined features. The inherent adaptability of the system contributes to its resilience against environmental variations and various forms of shoplifting behaviors. Additionally, the early identification of criminal intentions by the 3D CNN allows store personnel enough time for intervention and crime prevention. This novel approach has significant potential in enhancing security measures not only in retail establishments but also in other businesses, opening new possibilities for crime prevention strategies.

The discussed academic papers showcase a wide range of applications for machine learning and deep learning methods in predicting and preventing crimes. These studies underscore the significance of data preprocessing for improving model precision, selecting relevant features, and leveraging different algorithms including K-nearest neighbors, decision trees, support vector machines, and neural networks. Additionally, these research works emphasize the importance of accessible and trustworthy datasets while also addressing challenges related to inherent biases and comprehensibility in machine learning models. Several research papers concentrate on different facets of crime prediction, including the anticipation of crime types, criminal intentions, and initial stages of unlawful behavior

2. Model Development

2.1. Model Proposals

The aim of our project is to do a thorough analysis on crime data from the city of Chicago, model it by using various supervised classification algorithms and develop a framework which will correctly classify crime and propose safe neighborhood to live. Our dataset is collected from a reliable source the official website of the United States government. We have gathered crime data from the city of Chicago, specifically focusing on incidents from 2001 to the present. The dataset contains features like crime types, locations, and timestamps which is a solid base for analysis. Additionally, we have features like location description, arrest, district, ward, community area that can contribute valuable information to give a boost in improving the accuracy of our models. Our dataset is in a structured format which contains 21 features. As our target variable 'Crime_Type' encompasses of various crimes like theft, assault, homicide, narcotics, gambling, etc. we have decided to use multiclass classification algorithms. Saying that, we plan to employ both one-vs-all and one-vs-one strategy as mentioned in figure 43 and figure 44 in model development to make sure that we take all complexities into account and identify relevant patterns.

Figure 43

One-vs-One (OVO) approach for multiclass classification algorithm

One-vs-One (OvO):

Input:

N classes

C_i and C_j are two classes

Binary classifier trained for each pair of classes ($N \times (N-1)/2$ classifiers)

Output:

$$\text{Prediction} = \text{argmax}\left(\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \text{Votes}(C_i, C_j)\right)$$

Note. Mathematical representation of One-vs-One approach.

Figure 44

One-vs-All (OVA) approach for multiclass classification algorithm

One-vs-All (OvA):

Input:

N classes

C_i is a class

Binary classifier trained for each class to distinguish it from all other classes

Output:

$$\text{Prediction} = \text{argmax}(\text{Confidence}(C_i))$$

Note. Mathematical representation of One-vs-All approach.

As our dataset consists of diverse features, especially because we are working on a huge dataset consisting of 7M+ records, a probabilistic framework is needed for interpreting crime probabilities. Naive Bayes classifiers are highly efficient models for multiclass classification, making them particularly well-suited for base models. In research done by Bharati et. al (2018) Multinominal Naïve Bayes classifier gave an accuracy of 45.62%. In our case especially, the Multinomial Naive Bayes classifier is well-suited as it is known for efficiently handling categorical features. This will be a better model for us as it will handle scenarios where crimes can be categorized into distinct types based on various attributes such as location, time, etc. F Hermawan et. al (2023) suggest that KNN algorithm outperformed other classification algorithms with an accuracy of 94% at k=9. The author also mentions that from the results obtained the most common type of theft crime from 2019-2022 was theft. The study done by Ahishakiye et. al (2017) reviews various classification models and finalizes J48 decision tree classifier due to its adaptable which gave an accuracy of 94.25% in predicting crime categories, making it a reliable for crime prediction compared to other algorithms like SVM, Naive Bayes

classifier, etc. Four classification models namely Multinomial Naïve Bayes, Random Forest, Decision Tree, and KNN for model selection are suggested by Bharathi et. al (2018), drawing on their success in comparable studies. Using various ensemble techniques an ensemble model will be formed by combining predictions from the four individual models.

Decision Tree Classifier

A decision tree algorithm comes from the family of supervised learning algorithm which can handle both classification and regression tasks. A decision tree classifier is non-parametric in nature and is used for classification tasks. It has advantages like simplicity, interpretability, versatility, and also it selects features on its own. The prediction of target variable is done by traversing from the root node to a leaf node. The root node is at the top from which its leaf nodes emerge. A leaf node is a class label and each node from it gives some sort of decision based on the value of a specific feature. Ahishakiye et. al (2017) explains that decision trees use recursive partitioning which means the source dataset is partitioned into subsets based on attribute value tests. This makes a set of decision rules which is hierarchical in nature. This is all done by C4.5 algorithm which is an extension of Quinlan's earlier ID3 algorithm. In order to create subsets with as homogeneous class labels as possible, the algorithm recursively selects the best attribute to split the data into before building a decision tree. Building a tree that generalizes well to new, unseen data is the main motive of algorithm.

The calculation of entropy and information gain are the essential factors to build a decision tree. Entropy is a measure of impurity or disorder within a dataset, and it is done analyzing class labels distribution as given in equation 1.

(1) $$H(D) = -\sum_{i=1}^{c} p_i \cdot \log_2(p_i)$$

Where D is the given dataset, c is number of classes and p_i is proportion of instances in

class *i*.

Another important term is information gain which is the entropy reduction that comes from dataset splitting on a particular attribute as given in equation 2.

$$IG(D, A) = H(D) - \sum_{v \in \text{values}(A)} \frac{|D_v|}{|D|} \cdot H(D_v) \quad (2)$$

Where Information Gain is IG (D, A), D is dataset, A is an attribute, values(A) is the set of possible values for attribute A, D_v is the subset of D for which attribute A has value v.

Lastly, Gain Ratio is used to normalize the information gain. This is done by adding a bias towards attributes with a large number of values as given in equation 3.

$$\text{GainRatio}(D, A) = \frac{IG(D,A)}{-\sum_{v \in \text{values}(A)} \frac{|D_v|}{|D|} \cdot \log_2\left(\frac{|D_v|}{|D|}\right)} \quad (3)$$

Where Information Gain is IG (D, A), D is dataset, A is an attribute, values(A) is the set of possible values for attribute A, D_v is the subset of D for which attribute A has value v.

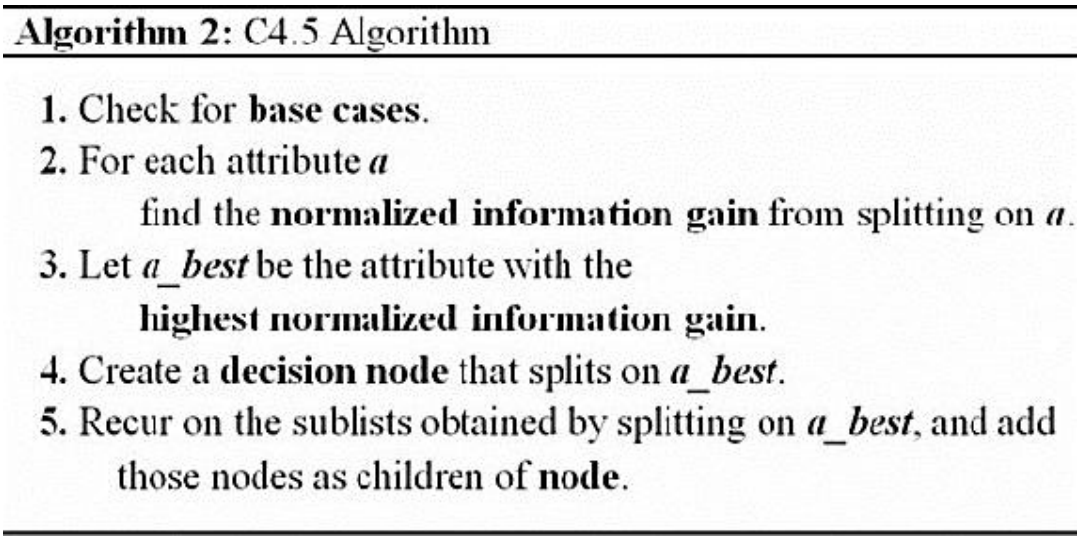
Where IG(D, A) is the Information Gain of attribute A in dataset D. $|D_v|$ is the number of instances in subset D_v ,values(A) is the set of possible values for attribute A, |D| is the total number of instances in dataset D.

The denominator is the Split Information which is attribute values entropy. The negative sign means the Gain Ratio is positive. The numerator is Information Gain.

The algorithm splits the dataset hierarchically by choosing the attribute that maximizes information gain and this process is recursive unless specific stopping conditions are satisfied. Stopping conditions can be anything from reaching a predetermined depth in the tree or having minimum instances in a node. The C4.5 algorithm contains cost-complexity pruning which helps to make tree generalize well capabilities by removing unnecessary branches. Lungu et. al (2010) explains the pseudocode as mentioned in Figure 45.

Figure 45

C4.5 Algorithm for decision tree



Algorithm 2: C4.5 Algorithm

1. Check for **base cases**.
2. For each attribute *a*
   find the **normalized information gain** from splitting on *a*.
3. Let *a_best* be the attribute with the
   highest normalized information gain.
4. Create a **decision node** that splits on *a_best*.
5. Recur on the sublists obtained by splitting on *a_best*, and add those nodes as children of **node**.

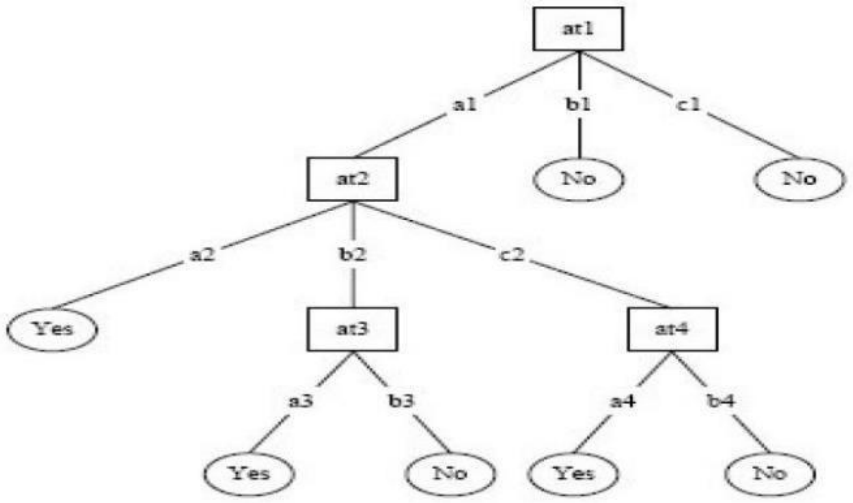
Note. A decision tree-based algorithm called C4.5 algorithm adapted from "Research issues concerning algorithms used for optimizing the data mining process by Lungu et. al (2010).

According to Lungu et. al (2010) C4.5 is a divide-and-conquer algorithm to create an initial tree. For recursive partitioning it uses attribute-based tests. These tests are ranked using information gain and gain ratio.

Overfitting is avoided by using a pruning algorithm that prunes the initial tree and estimates error rates. The algorithm has a list of rules for complex decision trees which makes them very easy to work with. These rule sets built from unpruned decision trees are made easy using hill-climbing algorithm. This helps in reduction of the pessimistic error rate.

Figure 46

Representation of Decision Tree



Note. Decision Tree Diagram adapted from "Crime Prediction Using Decision Tree (J48) Classification Algorithm by Ahishakiye et. al (2017).

As mentioned in Figure 46, Ahishakiye et. al (2017) mentions Decision trees do recursive splitting of data based on the most significant attribute at each node which leads to a decision at the end. There are various methods of splitting such as Gini impurity, information gain, entropy but during this splitting, overfitting can happen. Therefore, hyperparameter tuning is very important to optimize decision tree performance. Popular hyperparameters methods include max_depth and min_samples_split. By doing this the tree is generalized well on complex, new, unseen data.

Rokach et. al (2008) mentions that decision trees can be used for both classification and regression tasks. They have a structured framework and are called as decision-makers as developers can employ effective strategies to achieve their goals. Decision trees are popular because of their simplicity and transparency, thus making them accessible to a lay man. In classification tree, the recursive partitioning takes place. The rooted tree structure is the instance space and has internal nodes and leaves which are called test nodes and decision nodes respectively. Traversing from the root node to a leaf node based on attribute values is an intuitive

interpretation of how the decision tree classifies instances. An important feature of decision trees is they are not only used for nominal data but also for numeric attributes which makes them accessible for all types of datasets (p. 5 - 11).

Random Forest Classifier

Research by Bharati et al. (2018) suggests that employing machine learning techniques such as KNN classification, Logistic Regression, Decision trees, and Random Forest on the dataset of crime in Chicago can greatly enhance crime prediction and analysis. This could serve as a basis for improving law enforcement measures. According to a study conducted by Walczak et al., it has been found that Neural Network models, specifically single-hidden layer MLP, exhibit better accuracy in predicting the location of a crime compared to forecasting the type of crime. The researchers emphasize the significance of clustering as a means to address imbalances in different types of crimes. With the use of single-hidden layer MLP, they were able to achieve an accuracy rate of 31.2%. In conclusion, Random forests, Neural networks and decision trees are some of the choices made by the researchers.

Patel et. al (2022) says that Random Forest represents an ensemble learning technique which creates numerous decision trees and combines them to achieve a more reliable and precise prediction. The decision trees in Random Forest are often generated using the CART algorithm, which is a recursive binary splitting approach that partitions the data into subsets based on feature values. At each node, it identifies the optimal feature and split point to enhance class separation or minimize variance in the target variable.

As mentioned by Abdulraheem M. et. al (2022) Random Forest entails constructing numerous decision trees independently, with each utilizing a random portion of the training data and features at every split. This deliberate randomness results in diverse trees. The ultimate

prediction in Random Forest typically involves averaging or majority voting on individual tree predictions, producing a more resilient and precise model compared to a single decision tree.

Like Decision Tree, Random Forest also employs metrics like the Gini index or entropy as mentioned in equation 1, 2, 3 to determine how nodes should be split when creating trees in the ensemble. These metrics assess the purity of a node, evaluating how effectively a split separates classes in classification tasks. The Gini index is often utilized for classification purposes to minimize misclassification probability, while entropy measures information gain and prioritizes uncertainty reduction. At each node, the algorithm selects the split that maximizes purity or information gain. Through aggregating predictions from numerous trees, Random Forest enhances predictive precision and resilience in classification tasks.

The Random Forest is a highly efficient and effective supervised machine learning model, especially for classification tasks. It creates multiple decision trees that evaluate different subsets of data features independently and combines their predictions to achieve accurate results. This ensemble approach provides several advantages to the Random Forest algorithm in delivering reliable predictions. Random Forests use a voting mechanism where each decision tree in the ensemble contributes to making a prediction. By taking the majority vote among all trees, Random Forests reduce the risk of overfitting and improve predictive efficiency. This approach ensures that the final prediction is based on multiple models rather than one individual model, enhancing accuracy and reducing noise. Abdulraheem M. et al.'s study in the LAUTECH Journal of Engineering and Technology highlights the growing problem of crime in Nigeria and the limitations of reactive approaches. By utilizing machine learning, specifically the Random Forest algorithm, the authors propose a proactive strategy for identifying crimes. The ensemble learning aspect of this algorithm is beneficial for handling large datasets with an impressive

accuracy of 81.4%. When compared to other models, Random Forest demonstrates its effectiveness. This research has important implications including increased public awareness and better resource allocation for preventing crime. However, further investigation is recommended to fine-tune parameters and address any imbalances between classes within the data set. Overall, this literature emphasizes how Random Forest plays a significant role in predicting criminal activities and offers valuable insights into crime rate forecasting. Guo et al. conducted a study on applying different AI techniques to estimate mining capital costs in open-pit copper projects. The four techniques used were artificial neural network, random forest, support vector machine, and classification and regression tree. The Random Forest approach, first introduced by Breiman et al. is known for its effectiveness in predicting future outcomes. This algorithm uses a collection of decision trees and incorporates techniques such as bagging and random feature selection to improve accuracy. The research offers valuable insights into the application of AI techniques, particularly Random Forest, to optimize cost estimation in mining ventures. Figure 47 shows the pseudocode for the Random Forest algorithm as described in the book by Guo et al. (2021).

Figure 47

Random Forest Algorithm Pseudo code

Algorithm 1: Pseudo code for the random forest algorithm

To generate c classifiers:
for i = 1 to c **do**
  Randomly sample the training data D with replacement to produce D_i
  Create a root node, N_i containing D_i
  Call BuildTree(N_i)
end for

BuildTree(N):
if N contains instances of only one class **then**
  return
else
  Randomly select x% of the possible splitting features in N
  Select the feature F with the highest information gain to split on
  Create f child nodes of N, N_1,..., N_f, where F has f possible values (F_1, ... , F_f)
  for i = 1 to f **do**
    Set the contents of N_i to D_i, where D_i is all instances in N that match F_i
    Call BuildTree(N_i)
  end for
end if

Note. A pseudo code for random forest algorithm adapted from "Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach by Lungu et. al (2010).

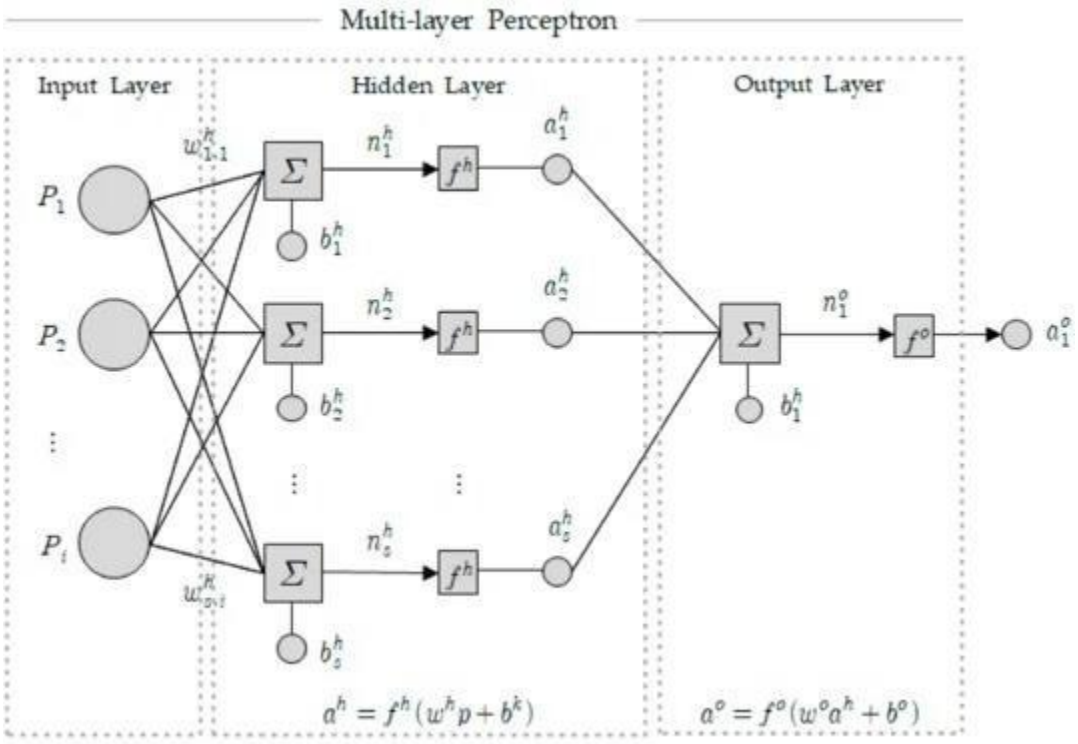
Artificial Neural Network

Artificial neural networks (ANNs) belong to the neural networks family and are one of popular machine learning models. ANNs are inspired by the working of the human brain. It has interconnected nodes which match neurons in the human brain. As these concepts are from machine learning technology, these interconnected nodes are known as artificial neurons. These artificial neurons transmit signals through edges, which connect multiple neurons. Each artificial neuron processes the signals received and sends signals to other connected neurons. The output of each neuron is nothing but a non-linear function which is applied to inputs sum. There is a concept called as signal, similar to biological synpases which is a connection is denoted by a real number. The ANNs also have a learning process and during this process the edges weights adjust

making signal strength stronger. There is also a concept called threshold which makes sure a signal is sent only when it surpasses that threshold.

Figure 48

Multi-layer perceptron diagram



Note. Multi-layer Percepton adapted from "Artificial Neural Network Model Development to Predict Theft Types in Consideration of Environmental Factors by Kwon et. al (2021).

Kwon et. al (2021) explains that neurons are organized into layers which perform unique transformations on their inputs. The signal flows from the first layer called the input layer and stops to the final layer which is called output layer as shown in Figure 48. During this flow, it traverses through multiple layers which are also known as Hidden Layers. Due to this layering combination it adjusts its weight on its own which helps artificial neural networks to learn and generalize well from data.

Kurban et al. (2012) says that ANNs are inspired by the structure of human brain and it

also has similar functionality. It is based on forward propagation of input data during process of training. In the training process, it tries to adjust weights and biases through backpropagation. This is how it learns to make predictions or classifications. The ANNs are popular in market as they have a great ability to generalize from labeled training data to new, unseen data. Training deep neural networks with multiple hidden layers is a concept of deep learning which works best in tasks like image and speech recognition.

In a neural network, a neuron denoted as (j) receives input ($p_j(t)$) from preceding neurons and comprises several essential elements. The state of the neuron is represented by the activation ($a_j(t)$), which relies on an integer time parameter. A potential threshold (θ_j) remains static unless adjusted through learning. The activation function (f) calculates the new activation at the subsequent time step (t+1) based on the current activation, threshold, and net input, following the equation 4.

$$a_j(t+1) = f(a_j(t), p_j(t), \theta_j) \tag{4}$$

The output function computes the output from activation and often simplifies to identity function as mentioned in equation 5.

$$o_j(t) = f_{\text{out}}(a_j(t)) \tag{5}$$

On the surface, the input neurons are responsible for receiving external input to the network, while the output neurons transmit processed information. The calculation of neuron input mentioned in equation 6 involves summing up the outputs from other neurons using weights and possibly introducing a bias term. This additional bias expands the flexibility in how the neuron responds to inputs.

$$p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j} \tag{6}$$

Hien et. al (2017) detail a method for training and choosing the most effective ANN structure for a specific task, as depicted in the accompanying diagram. This approach involves generating input datasets comprising all potential combinations of variables, training ANNs using different inputs and numbers of neurons, and iterating the training procedure repeatedly for each setup. The main goal is to determine the ANN structure that produces the highest test R^2 (coefficient of determination), a commonly utilized metric for evaluating predictive capability. The outer loop goes through different input combinations, while the middle loop adjusts the number of neurons in the network and the innermost loop repeats training to accommodate variations in this process. Each time the ANN is trained, the highest test R^2 is recorded. After completing this process, the best-performing ANN with the highest test R^2 across all configurations is stored. This results in an ANN-Storage library containing the best predicting ANN for each variable combination.

Figure 49

Pseudocode for ANN Algorithm

```
1:  procedure ANN (Input, Neurons, Repeat)
    Create input database
2:      Input ← Database with all possible variable combinations
    Train ANNs
3:      for Input = 1 to End of input do          ▷ Change inputs for every run
4:          for Neurons = 1 to 20 do              ▷ Increase neurons for every run
5:              for Repeat = 1 to 20 do           ▷ Repeat run 20 times
6:                  Train ANN
7:                  ANN-Storage ← save highest test R^2
8:              end for
9:          end for
10:         ANN-Storage ← Save best predicting ANN depending on inputs
11:     end for
12:     return ANN-Storage ▷ Library with best predicting ANN for every variable
                             combination
13: end procedure
```

Note. Pseudocode for ANN adapted from “BioTOOL—a Readily and Flexible Biogas Rate

Prediction Tool for End-users by Hien et. al (2017).

This method systematically investigates different neural network structures and input combinations, offering a systematic approach to identifying the most efficient configuration for a specific task. The focus on achieving the highest test R^2 guarantees the selection of ANNs that exhibit superior predictive precision. The ultimate result is an ANN-Storage library, serving as a comprehensive repository of top-performing ANNs customized for various input scenarios.
Figure 49 consists of initialization, forward propagation, backward propagation, and weights updating.

XGBoost

XGBoost, also known as eXtreme Gradient Boosting, is a highly adaptable and potent machine learning method recognized for its effectiveness in regression and classification assignments. It follows an ensemble learning strategy by constructing a sequence of decision trees that address errors from previous iterations. To prevent overfitting, XGBoost integrates both L1 and L2 regularization techniques along with tree pruning to manage the depth of individual trees. With its built-in cross-validation functionality, XGBoost simplifies the determination of optimal boosting rounds while also handling missing values during preprocessing. The algorithm's support for parallel processing enhances computational efficiency and offers feature importance scores for better interpretability. Widely utilized in competitions and real-world scenarios, XGBoost is available in various programming languages with an accessible API suitable for users at all skill levels.

Cherif et. al (2019) says that XGBoost is a popular supervised learning technique known for its efficacy in regression and classification both. The algorithm enhances an objective function, comprising of a loss function and a regularization term. The loss function, represented

by s, gauges the disparity between actual target values yi and predicted values for every example in the training data. Meanwhile, the regularization term, is linked to each tree within the ensemble to guard against overfitting as mentioned in equation 7.

$$\Omega(\theta) = \underbrace{\sum_{i=1}^{n} d(y_i, \widehat{y_i})}_{Loss} + \underbrace{\sum_{k=1}^{K} \beta(f_k)}_{regularization} \tag{7}$$

The term for controlling the minimum gain needed for further leaf node partition, an L2 regularization term on the weights of leaf nodes, and an L1 regularization term on the leaf weights are all part of the regularization process. The iterative optimization process involves constructing trees and adjusting their weights to minimize the overall objective function as mentioned in equation 8.

$$\beta(f_t) = \gamma T + \frac{1}{2}\left[\alpha \sum_{j=1}^{T} \mid c_j \mid + \lambda \sum_{j=1}^{T} c_j^2\right] \tag{8}$$

As mentioned by Chen et al. (2016) in his book XGBoost: A Scalable Tree Boosting System, he says XGBoost depends on a variety of important parameters to manage its training procedure, including the learning rate, maximum tree depth, subsample ratio, column subsample ratio per tree, minimum child weight, and others. It is essential to carefully adjust these parameters in order to achieve the best possible model performance. The algorithm uses a greedy method for choosing splits that maximize gain when constructing trees as mentioned in Figure 50.

Figure 50

Pseudocode 1 for XGboost Split finding

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Note. Pseudocode 1 for XGboost Split finding adapted from "XGBoost: A Scalable Tree Boosting System by Chen et al. (2016).

XGBoost's split-finding algorithm is a vital element in the XGBoost ensemble learning approach, specifically designed for building robust predictive models using decision trees. This algorithm considers various input parameters such as feature dimension (d), total number of features (m), and a regularization parameter (λ) when operating on a node's instance set (I). It employs a greedy strategy to identify the optimal split by iterating through each feature, sorting instances based on their feature values. Subsequently, it updates local sums of gradients and Hessians while iterating through each instance to calculate the splits' scores based on the sums of gradients(G) and Hessians(H).

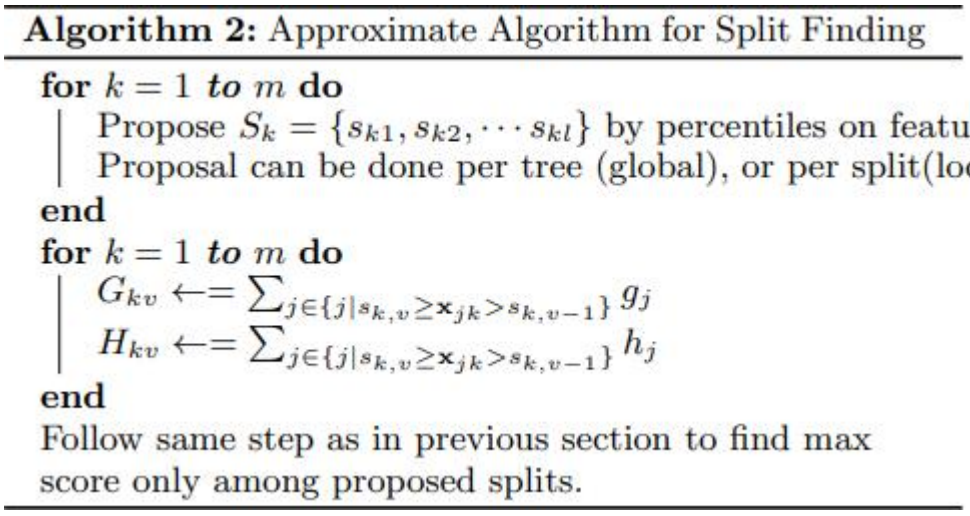
Chen et al. (2016) mentions that XGBoost utilizes a formula that includes the regularization term (λ) to balance model complexity and determine the split score. The algorithm aims to maximize this scoring function, selecting the split that produces the highest score for a specific feature. This process of finding splits is crucial within XGBoost's iterative method. With

each iteration, a new decision tree is built to handle the residuals of the combined model, continually improving predictions. The incorporation of a regularization term assists in preventing overfitting, thereby enhancing the model's ability to generalize.

The objective of the algorithm is to identify the most effective binary division for the current node by systematically evaluating all features (k) and instances (j). The split decision aims to maximize a scoring function that takes into account the sums of gradients and Hessians on both sides of the split, in addition to a regularization term (λ). This scoring function bears resemblance to those utilized within XGBoost. This algorithm forms an integral part of the procedure through which XGBoost constructs trees sequentially, with each tree serving to rectify errors from preceding ones. A greedy strategy is employed at each node to determine the optimal split, and this process iterates for a specified number of trees (m) or until meeting a stopping criterion.

Figure 51

Pseudocode 2 for XGboost Split finding



Note. Pseudocode 1 for XGboost Split finding adapted from "XGBoost: A Scalable Tree Boosting System by Chen et al. (2016).

Figure 51 describes an approximate method for identifying splits in the XGBoost framework. This method is particularly useful when a thorough search for optimal splits would be too computationally demanding. The algorithm consists of two primary loops. In the first loop, it generates a set of potential splits for each of the m features by dividing their values into percentiles. These proposed splits can be applied globally across the entire tree or locally at each split. In the second loop, the algorithm calculates gradients (g_j) and Hessians (h_j) for each proposed split. It computes G_{kv} and H_{kv}, which represent sums of gradients and Hessians respectively, for instance falling within intervals defined by each proposed split. This computation involves aggregating gradients and Hessians within individual intervals.

After acquiring the gradient and Hessian information for the suggested splits, the approach mirrors that outlined in figure 50. This entails executing identical procedures to identify the highest score among these proposed splits. The scoring mechanism probably incorporates a formula similar to that of the exact algorithm, taking into account Gkv, Hkv, and a regularization parameter (λ) aimed at preventing overfitting. The primary benefit of this approximate algorithm lies in its capacity to decrease computational complexity by focusing on only a subset of suggested splits instead of exhaustively evaluating all potential ones. The utilization of percentiles for proposing splits adds flexibility, enabling identification of pertinent points for dividing data distribution (p. 3, 4).

2.2. Model Supports

Environment, Platform, and Tools

For our project we are using Machine learning models to predict the Crime types in different areas. The main goal of this project is to ensure safer neighborhoods by predicting the Criminal behavior allowing for targeted interventions ahead of time, there by minimizing the

social and economic impact of criminal activities, including financial losses, community unrest, and emotional distress.

We have considered a dataset with more than 7 million records from the United States government website. Our dataset consists of historical data which includes 22 columns. As the dataset is about 1.86 GB, we have stored it in the google drive and have extracted it from there. We have used around 14GB of additional storage space in the process of building and tuning the models.

To implement our models, we decided to use Jupyter Notebooks and took advantage of the GPU capabilities available in Google Colab. Python was a natural choice for programming language due to its strong presence in the machine learning ecosystem. Our implementation involved constructing various models namely Random Forest, Decision Tree, Neural Networks, and XGBoost with the goal of thoroughly investigating and comparing their performance in predicting crimes. Given the sizeable dataset and intricate nature of machine learning algorithms, it became essential to utilize cloud-based resources. The inclusion of GPU support within Google Colab provided us with a flexible and scalable environment for developing our models. This configuration enabled us to exploit parallel processing capabilities which greatly expedited both training and evaluation processes.

To ensure the smooth execution of algorithms, a range of essential machine learning libraries and packages were utilized during the coding process. These included popular Python libraries such as pandas for data manipulation, NumPy for numerical operations, and matplotlib and seaborn for effective data visualization, sklearn is used for developing models. For selecting appropriate models in this study, various algorithms such as Random Forest, Decision Tree, Neural Networks, and XGBoost were implemented. Each model underwent fine-tuning and

optimization within the Jupyter Notebook environment while leveraging Colab's GPU acceleration to expedite training time. The code was structured to cater to the unique requirements of each algorithm with an emphasis on accurate predictions and efficient processing of extensive datasets. For building our 4 models sklearn library was used. Table 6 includes all the modules from sklearn library.

Table 6

Module, their functions, and their usage

Module	**Function**	**Used for**
sklearn.model_selection	train_test_split	Splitting the Dataset into Training, Testing and Validation sets in the ration of 70:20:10 respectively.
sklearn.ensemble	RandomForestClassifier	Implementing Random forest classifier.
sklearn.tree	DecisionTreeClassifier	Implementing Decision Tree classifier.
sklearn.neural_network	MLPClassifier	Implementing Neural Network classifier.
xgboost	XGBClassifier	Implementing XGBoost classifier.

Note. Table depicting Module, their functions, and their usage.

Model Architecture and Dataflow

The entire dataset is divided into 3 parts.70% of the dataset is considered as training dataset, 20% is considered as Validation dataset and 10% of the dataset is considered to be testing dataset.

As for the hyperparameter tuning which combines different parameters in an attempt to increase the performance of the models. For random forest, 'n_estimators' determines number of decision trees in each forest, 'max_depth' which determines the depth of each tree,

‘min_samples_split' determines number of samples required to split the internal node and 'min_samples_leaf' sets the minimum number of samples required by each leaf node. For Decision tree, ‘criterion’ defines the measure of quality of each split, ‘max_features’ confirms that all features are used in each split and the rest of the parameters used are ‘max_depth’, ‘min_samples_split’, min_samples_leaf which are similar to the parameters used in Random Forest.

TensorFlow and Keras implementation is used to construct a neural network model. The structure of the model involves multiple dense (fully connected) layers, with the input layer having dimensions equivalent to the features in the training dataset (X_train.shape). Subsequent hidden layers comprise 64 neurons each employing rectified linear unit activation for introducing non-linearity into the model. This pattern is repeated five times to create a deeper representation. The output layer consists of neurons equaling the number of classes (num_classes) in the target variable, utilizing softmax activation function for generating probability scores for each class. The model is constructed with the Adam optimizer, an adaptive learning rate optimization algorithm. The selection of the categorical crossentropy loss function is based on its appropriateness for handling multiclass classification problems, while monitoring the accuracy metric during training enables assessment of the model's performance.

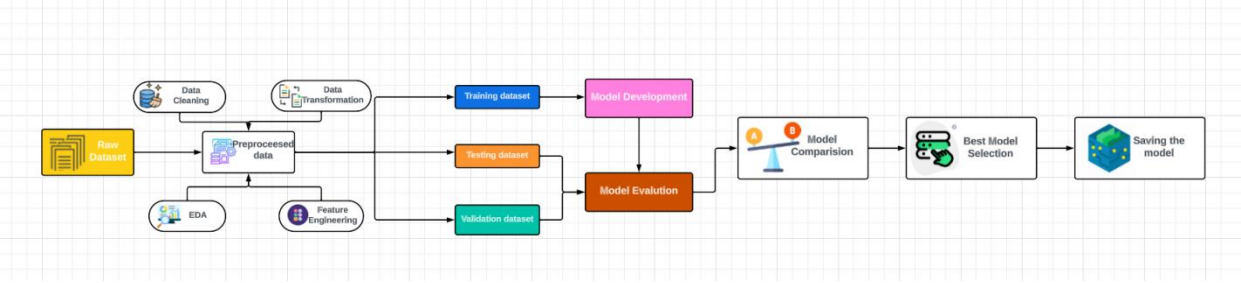
The XGBoost classifier is chosen for multi-class classification tasks and utilizes the 'multi:softmax' objective function to enhance prediction accuracy. By employing 'mlogloss' as the evaluation metric, it measures the alignment of predicted probabilities with the actual class labels. Various important hyperparameters such as 'n_estimators' for determining the number of trees, 'max_depth' to manage tree depth, and a crucial parameter like 'learning_rate', which controls iteration step size, contribute significantly to shaping this model's architecture.

Additionally, setting '-1' for 'n_jobs' enables parallel processing by utilizing all available cores effectively. It is worth noting that meticulous optimization of these parameters plays a pivotal role in achieving superior model performance, establishing XGBoost as an adaptable and extensively utilized tool in machine learning applications.

Figure 52 shows the architecture of the entire project. It starts with Collection of raw data and preprocessing the raw data by data cleaning, EDA, data transformation and Feature selection. Finally dividing the preprocessed dataset into 3 parts namely Training, Testing and validation. The training dataset is used in Model development where as testing and validation datasets are used for model evaluation. After evaluating all the models, models are compared based on their respective performance metrics, the model which has outperformed the rest of models is considered as the best model, which in our project is Decision Tree. The second-best model is Random Forest followed by the XGBoost and ANN models. We selected Decision Tree instead of Random Forest because Decision Tree is more interpretable and easier to visualize than Random Forest also Random Forest is computationally intensive and takes more time for training when compared to decision tree. The decision tree model was also saved using the joblib library, so that it can be used to make predictions on new data in the future.

Figure 52

Project Architecture



Note. An architecture diagram of entire project.

2.3. Model Comparison and Justification

Model Justification

Random forest is a popular machine learning method, which is proven to be a success in various problems. It is suitable for multiple classification, which is one of the essential requirements of our crime-type predictive model selection. When we train the random forest model on our large crime records, the predictions from different trees will be combined and voted. This ensemble nature introduces diversity and mitigates overfitting. Additionally, the architecture that allows building multiple trees simultaneously guarantees computational efficiency.

We aim to implement a decision tree model using the C4.5 algorithm. It effortlessly tackles both categorical and numerical variables and helps alleviate the burden of data preprocessing. The most attractive characteristic of the decision tree among all the other tree-based models is its interpretability. C4.5 algorithm decision tree employs a greed search using the criterion called information gain, exploring all the possible tree constructions to filter the optimal result. We can see how the entire tree is constructed and why we get such a prediction with a tool like 'graphviz' to visualize the tree. Its transparent mechanism makes the prediction reasonable and compelling.

XGboost is a widely known algorithm for its exceptional performance. We've tried both XGboost and Gradient Boost at the beginning of model development. As the optimization of Gradient Boost, the XGboost algorithm exhibits its strengths in the following aspects: It requires shorter training times, while Gradient Boost even cannot achieve the running outputs due to resource limitation. Builted in regularizations is another plus in addition to the computation cost, as overfitting is always a critical concern in our crime type prediction problem and other real-

world scenarios. What's more, our dataset displays significant data imbalance, which makes the model biased towards the majority classes to some extent. XGboost includes some settings to reduce the impact of data imbalance.

Artificial Neural Network is a deep learning method. Abundant existing works show its advancement in the crime prediction field. Aldossari et al.(2022) mentioned they got accuracy for crime type prediction at 99%. Crimes typically involve intricate patterns affected by various factors. ANN shows proficiency in capturing the non-linear relationship with data and usually achieves higher performance as the data size increases. Our dataset, which comprises more than 7 million records, is well-suited for leveraging the capabilities of ANNs. Additionally, ANN can automatically identify the relevant features during the learning process, addressing the challenges in manual feature engineering to obtain satisfactory accuracy.

Model Comparison

Table 7

Model Comparison stating Advantages and Disadvantages

Model	Advantages	Disadvantages
Random Forest	• Ensemble method contributes to model robustness • Different trees can be built simultaneously • Provides high accuracy • Works with both categorical and numerical variables	• Less Interpretable than DT • Computationally expensive on large dataset • Biased toward the dominant class
Decision Tree （C4.5）	• Provides human-readable prediction • Adapt to both categorical and numerical attributes • Can handle missing values • Scalable to large datasets	• Sensitive to noisy data and outliers • Designed for classification, limit support for regression • Easily lead to overfitting

XGBoost	• High accuracy and performance on various datasets • Faster than traditional algorithm using parallel processing • Includes regularization terms to reduce overfitting • Handling data imbalance and missing values	• Hard to find the optimal set of hyperparameters • Computationally expensive, especially for large datasets • Require large amount of memory • Sensitive to outliers and noises
ANN	• Achieves high accuracy by learning autonomously from data • Ability to capture the complex patterns of data • Can handle incomplete or noisy data • Generalizes well to unseen data	• Require large amount of data for training • Computationally intensive for deep neural networks • Model performance affected by data preprocessing • Black box nature makes it hard to interpret

Note. Table of Advantages and Disadvantages for each model.

To address our targeted problem, which is a multiple classification with a large imbalance dataset, we aim to apply various approaches, including traditional machine learning and trendy deep learning methods. The above Table 6 provides a summary of the model we proposed. Each model has their own strengths and weaknesses. For example, certain models exhibit no preference for dataset size, demonstrating versatility across small and large datasets, while other models are specifically designed to perform well with either small or large datasets. Furthermore, some models possess the capability to handle imbalanced data, while others may yield misleading performance metrics in the presence of data imbalance. Additionally, there is variability among these models in terms of their adaptability to different dimensionalities. Each model has the potential to address one or more aspects of our specific crime prediction task. We will conduct a performance comparison and retain the model that performs the best.

2.4. Model Evaluation Methods

We have evaluated the effectiveness of our models by using important metrics such as

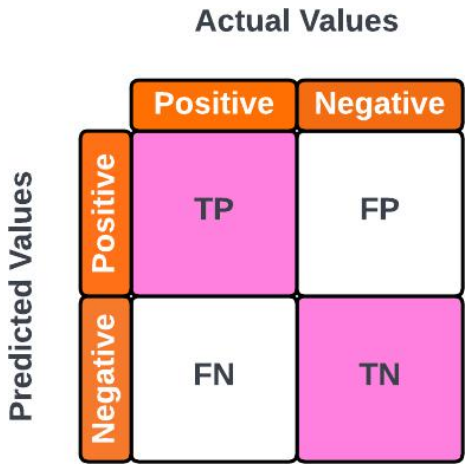
Accuracy, F1-score, Recall, and Precision. In addition to these quantitative measures, we also gain valuable insights from visual analysis of the ROC curve and a thorough examination of class-based evaluation through interpreting the Confusion matrix.

2.4.1. Confusion Matrix

The confusion matrix is an essential tool for assessing the accuracy of classification models. It consists of rows representing predicted labels and columns representing actual class labels. In binary classification, "True Positives" are instances correctly labeled as positive, while "False Positives" are instances wrongly labeled as positive when they belong to the negative class. Conversely, "False Negatives" indicate instances incorrectly classified as negative despite belonging to the positive class, and "True Negatives" represent accurately predicted negative instances. This matrix provides a detailed breakdown of model performance and helps identify and quantify prediction errors across classes as mentioned in Figure 53.

Figure 53

Confusion Matrix



Note. Confusion Matrix.

2.4.2. Accuracy

Accuracy is a fundamental evaluation metric used to assess the effectiveness of a machine learning model in classification tasks. It quantifies the ratio of correctly predicted instances to the total number of instances in the dataset as mentioned in equation 9.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

2.4.3. Recall

Recall, also known as sensitivity or true positive rate, evaluates the capability of a model to accurately identify all pertinent instances. It quantifies the proportion of correctly predicted positives out of the total number of actual positive cases in the dataset. It is calculated as the ratio of true positives to the sum of true positives and false negatives as mentioned in equation 10.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{True Negative}} (10)$$

2.4.4. Precision

This metric measures the proportion of correctly predicted positive instances out of all predicted positive instances, specifically emphasizing the accuracy of the model's positive predictions. It is computed by dividing the number of true positives by the sum of true positives and false positives as mentioned in equation 11.

$$\text{Precision} = \frac{}{+} \quad (11)$$

2.4.5. F-1 Score

It is a comprehensive measure that considers both precision and recall, providing a consolidated evaluation of the classifier's ability to balance between minimizing false positives and false negatives in order to reflect the model's performance as mentioned in equation 12.

$$\text{F1-score} = 2 * \frac{Precision * Recall}{+} \quad (12)$$

2.4.6. ROC (Receiver Operating Characteristic) and AUC (Area Under Curve):

The ROC curve demonstrates how the true positive rate (sensitivity) and false positive rate (1 - specificity) vary with changes in the discrimination threshold. It plots the TPR on the y-axis against FPR on the x-axis, where TPR represents correctly predicted positive instances relative to total actual positives, and FPR represents incorrectly predicted negative instances relative to total actual negatives.

As the threshold for discrimination changes, the ROC curve illustrates how the sensitivity and specificity of the model are affected. Ideally, a perfect classifier would have an ROC curve that reaches the top-left corner, indicating high true positive rate and low false positive rate across all thresholds. On the other hand, a diagonal line from bottom-left to top-right represents a random classifier with no predictive capability.

The AUC-ROC (area under the ROC curve) is a widely used metric for evaluating classifier performance. It measures the ability of the model to discriminate between positive and negative instances across different threshold settings. Higher values of AUC-ROC, nearing 1, indicate better discrimination ability and overall model performance. Conversely, an AUC-ROC value of 0.5 suggests that the model performs no better than random chance in its predictions.

The use of ROC curve and AUC-ROC is advantageous when comparing the performance of different models, especially in binary classification tasks. These metrics offer valuable insights into a model's capability to differentiate between classes and aid in selecting an optimal threshold by considering the trade-offs between true positive and false positive rates. This selection process can be customized according to specific application requirements.

2.5. Model Validation and Evaluation

Decision Tree

Baseline Model. DecisionTreeClassifier with default hyperparameter is used as a

baseline model for Decision Tree. We have used a Python library called Scikit-learn for our model development. The DecisionTreeClassifier was built on default hyperparameters like the 'gini' criterion which was used to assess split quality based on Gini impurity, the 'best' splitter strategy is used keeping in mind the most favorable split at each node. There is no limit on the growth of the tree that's why max_depth is set to None, allowing nodes to expand. min_samples_split is set to 2 which is default until the nodes contain fewer than the default of 2 samples. The minimum samples at a leaf node which is called min_samples_leaf is set to 1. The minimum weighted fraction set for a leaf node is 0.0 which means it's not necessary to have any minimum weighted fraction. Max features are set to None which means all features are considered for splitting at each node. The random_state parameter is also set to None (default) meaning it gives permission for non-deterministic behavior. The baseline model with default hyperparameters achieves an F1 score of 0.928 as shown in Figure 54. The ROC curve for decision tree given in Figure 55 was a bit rough. Training time for this model is roughly 62.69 seconds.

Figure 54

Classification report for Decision Tree Baseline Model

Classification Report:

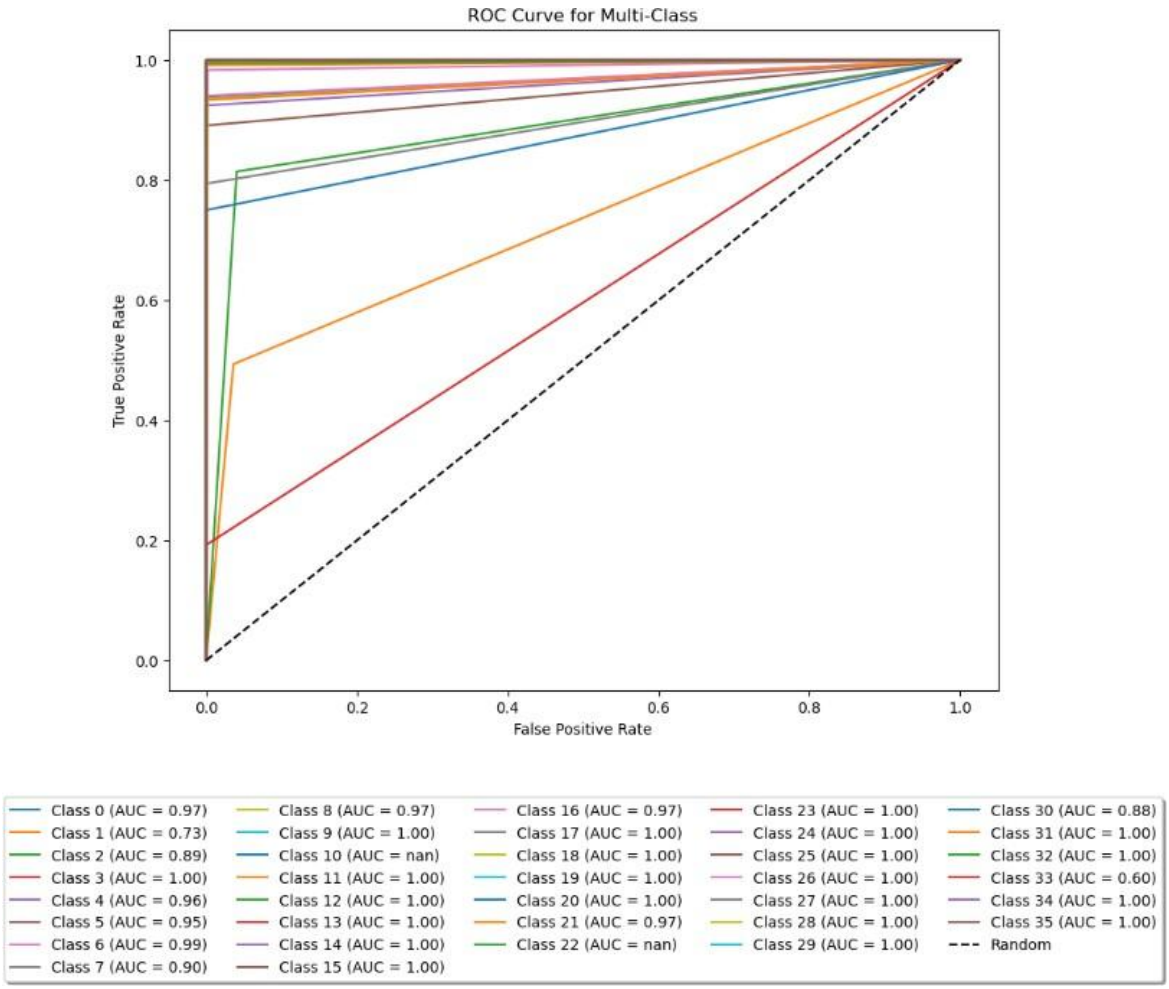
	precision	recall	f1-score	support
0	0.95	0.94	0.94	2683
1	0.49	0.49	0.49	104630
2	0.82	0.81	0.82	290171
3	1.00	1.00	1.00	86378
4	0.96	0.93	0.94	238
5	0.88	0.89	0.89	5696
6	0.98	0.98	0.98	181609
7	0.80	0.79	0.80	1579
8	0.93	0.94	0.93	43429
9	1.00	1.00	1.00	70841
11	0.99	0.99	0.99	2915
12	1.00	1.00	1.00	2583
13	1.00	1.00	1.00	25
14	1.00	1.00	1.00	3696
15	1.00	1.00	1.00	935
16	0.94	0.94	0.94	1462
17	1.00	1.00	1.00	3001
18	1.00	1.00	1.00	78306
19	1.00	1.00	1.00	150383
20	1.00	1.00	1.00	7
21	1.00	0.96	0.98	45
23	0.99	1.00	1.00	171
24	1.00	1.00	1.00	11375
25	1.00	1.00	1.00	29
26	1.00	1.00	1.00	98843
27	1.00	1.00	1.00	14240
28	1.00	1.00	1.00	38
29	1.00	1.00	1.00	10502
30	1.00	0.75	0.86	4
31	1.00	1.00	1.00	59699
32	0.99	1.00	0.99	6429
33	0.18	0.19	0.19	1048
34	1.00	1.00	1.00	335975
35	1.00	1.00	1.00	22254
accuracy			0.93	1591219
macro avg	0.94	0.93	0.93	1591219
weighted avg	0.93	0.93	0.93	1591219

Accuracy Score: 92.77%
Precision: 0.928
Recall: 0.928
F1 Score: 0.928

Note. Decision Tree Classification Report for Baseline Model.

Figure 55

AUC-ROC curve for Decision Tree



Note. Decision Tree AUC-ROC Curve for Baseline Model.

Hyperparameter Tuned Model. We use GridSearchCV to do a grid search to find out the best model along with the best hyperparameters for this dataset. It took approximately 8 hours to do an extensive grid search using various hyperparameters like criterion, maximum depth, minimum samples split, minimum samples leaf and maximum features.

After running grid search, the criterion is set to ‘entropy’, maximum depth is set to 15, maximum features are set to None, minimum samples leaf is set to 4 and minimum samples split is set to 5. By making predictions and calculating accuracies we have evaluated the model's performance. After getting the expected performance from the validation set, the model is

assessed on an independent test set. By doing this we determine how well the trained decision tree generalizes to new, unseen data. This will help in providing insights into its overall effectiveness and reliability. The hyperparameter tuned model achieves an F1 score of 0.94 which is slightly higher than the accuracy of the baseline model with default hyperparameters as shown in Figure 56. The ROC curve for multiclass is shown in Figure 57 which is smoother than baseline model. Training time for this model is roughly 65.13 seconds which is significantly larger than the training time of the baseline model with default hyperparameters.

Figure 56

Classification report for Decision Tree Hyperparameter tuned model.

Classification Report:

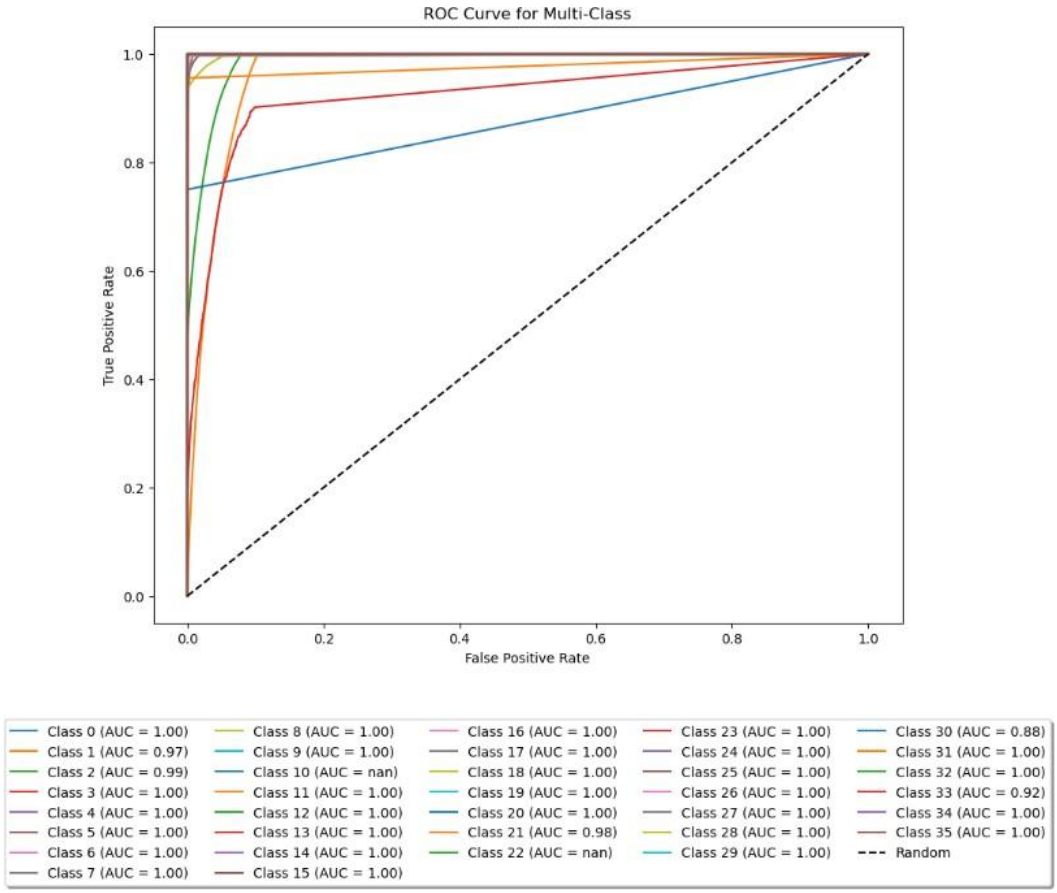
	precision	recall	f1-score	support
0	0.94	0.97	0.96	2683
1	0.62	0.46	0.52	104630
2	0.82	0.90	0.86	290171
3	1.00	1.00	1.00	86378
4	0.97	0.96	0.96	238
5	0.95	0.92	0.93	5696
6	0.98	1.00	0.99	181609
7	0.96	0.74	0.84	1579
8	1.00	0.93	0.96	43429
9	1.00	1.00	1.00	70841
11	0.99	0.99	0.99	2915
12	1.00	1.00	1.00	2583
13	1.00	1.00	1.00	25
14	1.00	1.00	1.00	3696
15	1.00	1.00	1.00	935
16	1.00	0.94	0.97	1462
17	1.00	1.00	1.00	3001
18	1.00	1.00	1.00	78306
19	1.00	1.00	1.00	150383
20	1.00	1.00	1.00	7
21	0.98	0.96	0.97	45
23	0.99	1.00	1.00	171
24	1.00	1.00	1.00	11375
25	1.00	1.00	1.00	29
26	1.00	1.00	1.00	98843
27	1.00	1.00	1.00	14240
28	1.00	1.00	1.00	38
29	1.00	1.00	1.00	10502
30	1.00	0.75	0.86	4
31	1.00	1.00	1.00	59699
32	1.00	0.99	1.00	6429
33	1.00	0.18	0.31	1048
34	1.00	1.00	1.00	335975
35	1.00	1.00	1.00	22254
accuracy			0.94	1591219
macro avg	0.98	0.93	0.94	1591219
weighted avg	0.94	0.94	0.94	1591219

Accuracy Score: 94.28%

Note. Decision Tree Classification Report for Hyperparameter tuned model.

Figure 57

AUC-ROC curve for hyperparameter tuned model.



Note. Decision Tree AUC-ROC curve for hyperparameter tuned model.

Random Forest

Baseline Model. The baseline Random Forest model is created using the RandomForestClassifier from Scikit-learn, utilizing 150 decision trees. Essential parameters like 'max_depth,' 'min_samples_split,' and 'min_samples_leaf' are adjusted to improve model effectiveness. It undergoes fitting on a training dataset with 6 features and 36 crime labels for predictive accuracy assessment. The inclusion of the 'random_state' parameter ensures reproducible results. Post-fitting, the model's accuracy and performance metrics are evaluated. This Random Forest model offers a reliable machine learning method for classification tasks by harnessing an ensemble of decision trees to enhance prediction capabilities. The accuracies and

f1 scores for validation and testing datasets which are 74.5% and 70.3%, 74.5% and 70.2% respectively. Figure 58 and Figure 59 show the classification report of the validation dataset and testing dataset respectively where f1 score, precision, recall and support are calculated.

Confusion matrix of Validation dataset is displayed as Figure 60 and the confusion matrix for the Test dataset is displayed as Figure 61. Figure 62 is the ROC plot of baseline model.

After evaluation of the model, the performance of the model is satisfactory, However, hyperparameter tuning is implemented in an attempt to increase the performance.

Figure 58

Classification Reports Of Validation Datasets Of The Baseline Model

Classification Report of Validation dataset on Baseline model:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2752
1	0.65	0.03	0.06	104373
2	0.60	0.97	0.75	290987
3	0.94	0.83	0.88	85657
4	0.00	0.00	0.00	220
5	0.00	0.00	0.00	5547
6	0.94	0.96	0.95	181291
7	0.00	0.00	0.00	1576
8	1.00	0.76	0.86	43262
9	0.78	0.39	0.53	71180
11	0.00	0.00	0.00	2872
12	0.00	0.00	0.00	2652
13	0.00	0.00	0.00	22
14	0.00	0.00	0.00	3776
15	0.00	0.00	0.00	999
16	1.00	0.12	0.21	1491
17	0.00	0.00	0.00	2974
18	0.83	0.79	0.81	78765
19	0.76	0.77	0.76	151049
20	0.00	0.00	0.00	8
21	0.00	0.00	0.00	41
22	0.00	0.00	0.00	1
23	0.00	0.00	0.00	197
24	0.74	0.01	0.02	11398
25	0.00	0.00	0.00	34
26	0.88	0.44	0.58	98878
27	0.86	0.49	0.62	14053
28	0.00	0.00	0.00	38
29	0.00	0.00	0.00	10632
30	0.00	0.00	0.00	3
31	0.95	0.41	0.57	60413
32	0.00	0.00	0.00	6308
33	0.00	0.00	0.00	1023
34	0.70	0.96	0.81	336442
35	0.95	0.87	0.91	22298
accuracy			0.75	1593212
macro avg	0.36	0.25	0.27	1593212
weighted avg	0.75	0.75	0.70	1593212

Note. F1 scores and accuracies of the baseline model Validation datasets.

Figure 59

Classification Reports Of Test Datasets Of The Baseline Model

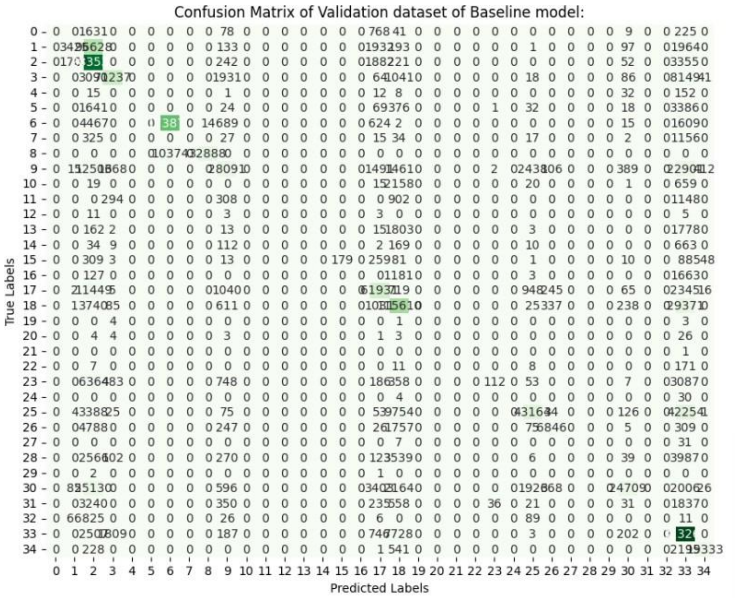
Classification Report of Testing dataset of Baseline model:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1281
1	0.64	0.03	0.06	51657
2	0.60	0.97	0.75	143368
3	0.95	0.83	0.89	42516
4	0.00	0.00	0.00	103
5	0.00	0.00	0.00	2775
6	0.94	0.96	0.95	89570
7	0.00	0.00	0.00	752
8	1.00	0.76	0.86	21757
9	0.78	0.39	0.52	34693
11	0.00	0.00	0.00	1504
12	0.00	0.00	0.00	1255
13	0.00	0.00	0.00	10
14	0.00	0.00	0.00	1860
15	0.00	0.00	0.00	451
16	1.00	0.13	0.22	719
17	0.00	0.00	0.00	1514
18	0.83	0.78	0.81	38549
19	0.76	0.77	0.76	74439
20	0.00	0.00	0.00	3
21	0.00	0.00	0.00	15
22	0.00	0.00	0.00	1
23	0.00	0.00	0.00	67
24	0.74	0.01	0.02	5654
25	0.00	0.00	0.00	16
26	0.88	0.44	0.58	48629
27	0.85	0.48	0.61	6908
28	0.00	0.00	0.00	20
29	0.00	0.00	0.00	5217
30	0.00	0.00	0.00	3
31	0.94	0.41	0.57	29495
32	0.00	0.00	0.00	3131
33	0.00	0.00	0.00	503
34	0.70	0.96	0.81	165336
35	0.95	0.87	0.91	10946
accuracy			0.75	784717
macro avg	0.36	0.25	0.27	784717
weighted avg	0.75	0.75	0.70	784717

Note. F1 scores and accuracies of the baseline model Testing datasets.

Figure 60

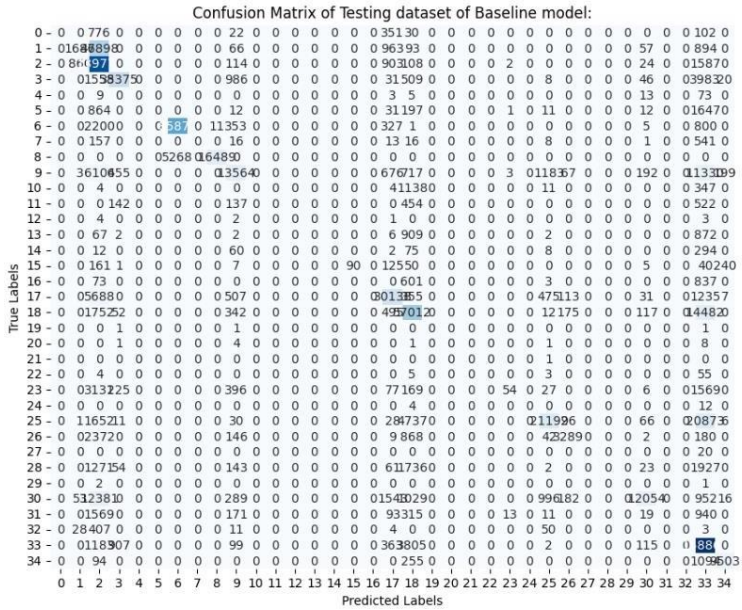
Confusion matrix of Validation dataset of Baseline model



Note. Confusion matrix of Validation dataset of Baseline model.

Figure 61

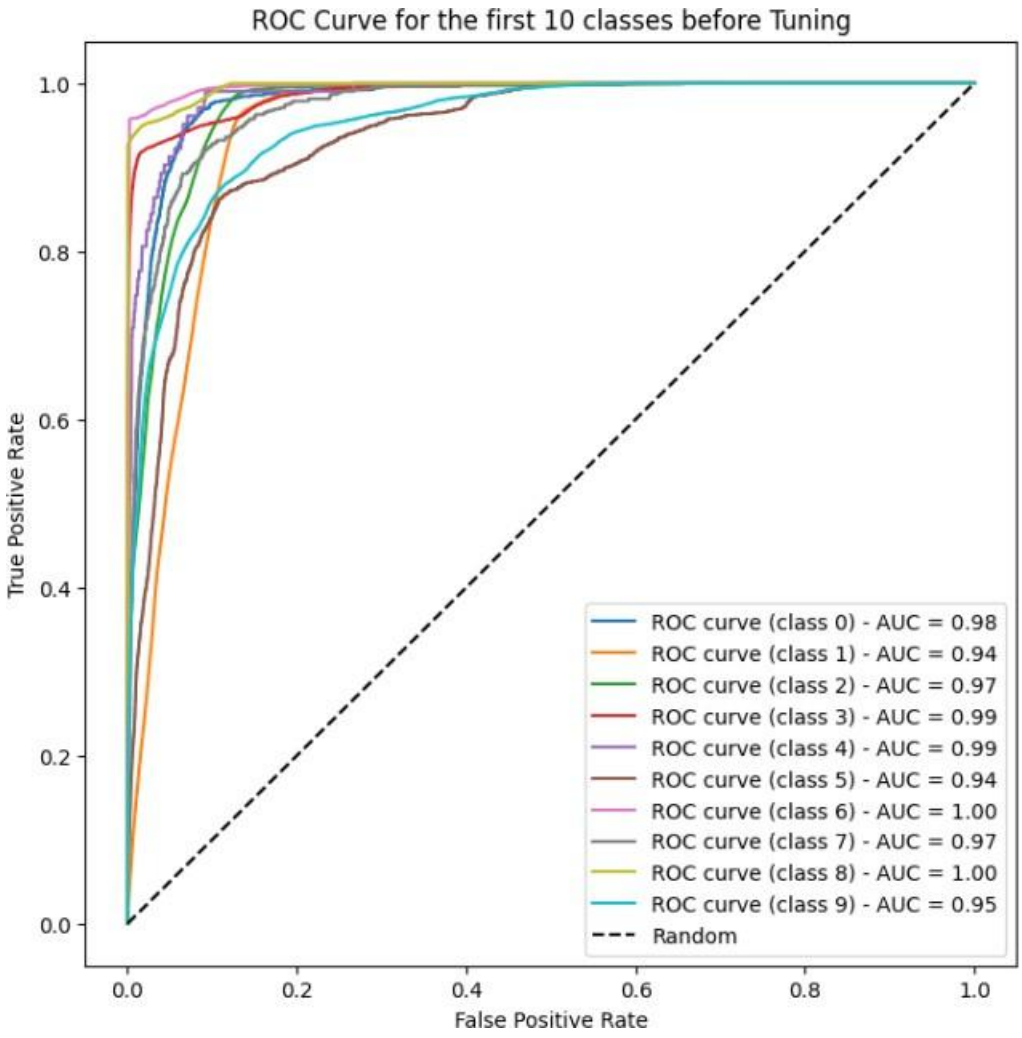
Confusion matrix of Testing dataset of Baseline model



Note. Confusion matrix of Validation dataset of Test model.

Figure 62

ROC plot of baseline model



Note. ROC plot of baseline model.

Hyperparameter tuned Model. A grid search was performed to discover the best hyperparameters. The improved model utilizes 50 decision trees and carefully adjusted parameters such as 'min_samples_split,' 'min_samples_leaf,' and 'max_depth.' This meticulous adjustment significantly enhances the accuracy of the model, resulting in a validation accuracies of 92.96% and a test accuracy of 92.98%, F1 score of validation dataset is 92.74% and F1 score of the Test dataset is 92.76%. Before hyperparameter tuning, the initial model showed lower validation and test accuracies at 72.97% and 73.05% and F1 scores of 70.3% and 70.2% ,

respectively. The enhancement achieved through hyperparameter tuning highlights its effectiveness in optimizing the performance of the Random Forest classifier. Figure 63 displays the classification reports of Validation and test datasets of the hyper parameter tuned model.

Figure 64 is the confusion matrix of Validation dataset of Hyper parameter tuned model, and figure 65 and figure 66 is the confusion matrix of validation and testing dataset. Figure 67 shows the ROC curve of the Hyper tuned model which is very smooth when compared to the ROC curve of the Baseline model.

Figure 63

Classification Reports Of Validation Datasets Of The Hyper parameter tuned Model

Classification Report of Validation dataset of Hyper parameter Model:

	precision	recall	f1-score	support
0	0.90	0.78	0.83	2752
1	0.56	0.47	0.51	104373
2	0.82	0.87	0.84	290987
3	0.99	1.00	1.00	85657
4	0.94	0.62	0.75	220
5	0.90	0.75	0.82	5547
6	0.98	1.00	0.99	181291
7	0.82	0.57	0.67	1576
8	1.00	0.93	0.96	43262
9	0.97	0.98	0.98	71180
11	0.97	0.92	0.95	2872
12	0.97	0.97	0.97	2652
13	1.00	0.09	0.17	22
14	0.93	0.82	0.87	3776
15	0.85	0.44	0.58	999
16	0.91	0.68	0.78	1491
17	0.89	0.85	0.87	2974
18	0.99	1.00	1.00	78765
19	0.99	0.99	0.99	151049
20	1.00	0.12	0.22	8
21	0.67	0.05	0.09	41
22	0.00	0.00	0.00	1
23	0.86	0.10	0.17	197
24	0.95	0.88	0.92	11398
25	0.50	0.12	0.19	34
26	0.97	0.99	0.98	98878
27	1.00	0.98	0.99	14053
28	1.00	0.05	0.10	38
29	0.95	0.86	0.90	10632
30	0.00	0.00	0.00	3
31	0.98	0.99	0.99	60413
32	0.93	0.81	0.87	6308
33	0.58	0.08	0.14	1023
34	1.00	1.00	1.00	336442
35	0.99	0.98	0.99	22298
accuracy			0.93	1593212
macro avg	0.85	0.65	0.69	1593212
weighted avg	0.93	0.93	0.93	1593212

Note. Classification Reports Of Validation Datasets Of The Hyper parameter tuned Model.

Figure 64

Classification Reports Of Test Datasets Of The Hyper parameter tuned Model

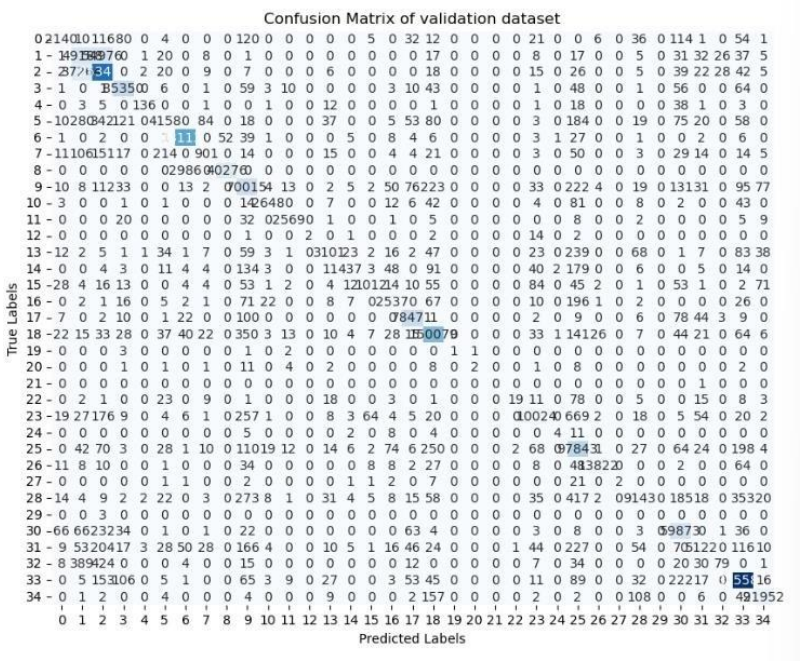
Classification Report of Test dataset of Hyper parameter Model:

	precision	recall	f1-score	support
0	0.90	0.77	0.83	1281
1	0.57	0.48	0.52	51657
2	0.82	0.87	0.84	143368
3	0.99	1.00	0.99	42516
4	0.88	0.56	0.69	103
5	0.89	0.74	0.81	2775
6	0.98	1.00	0.99	89570
7	0.81	0.54	0.65	752
8	1.00	0.93	0.96	21757
9	0.97	0.98	0.98	34693
11	0.97	0.92	0.94	1504
12	0.97	0.97	0.97	1255
13	0.00	0.00	0.00	10
14	0.93	0.81	0.87	1860
15	0.83	0.44	0.57	451
16	0.91	0.69	0.78	719
17	0.90	0.85	0.87	1514
18	0.99	1.00	1.00	38549
19	0.99	0.99	0.99	74439
20	0.00	0.00	0.00	3
21	1.00	0.33	0.50	15
22	0.00	0.00	0.00	1
23	0.71	0.07	0.14	67
24	0.95	0.88	0.91	5654
25	1.00	0.12	0.22	16
26	0.97	0.99	0.98	48629
27	1.00	0.98	0.99	6908
28	1.00	0.10	0.18	20
29	0.96	0.86	0.90	5217
30	0.00	0.00	0.00	3
31	0.98	0.99	0.98	29495
32	0.94	0.81	0.87	3131
33	0.64	0.10	0.17	503
34	1.00	1.00	1.00	165336
35	0.99	0.99	0.99	10946
accuracy			0.93	784717
macro avg	0.81	0.65	0.69	784717
weighted avg	0.93	0.93	0.93	784717

Note. Classification Reports Of Test Datasets Of The Hyper parameter tuned Model.

Figure 65

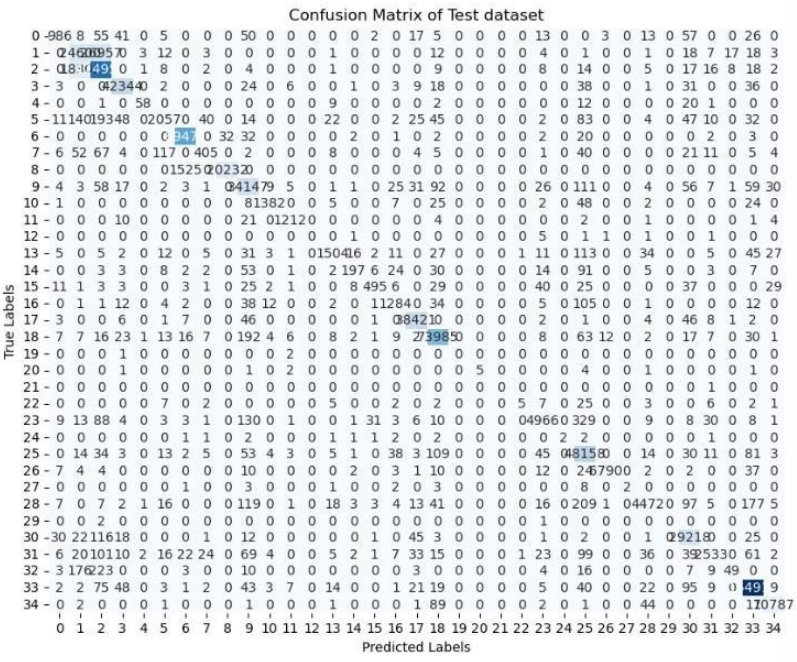
Confusion matrix of Validation dataset of Hyper parameter tuned model



Note. Confusion matrix of Validation dataset of Hyper parameter tuned model.

Figure 66

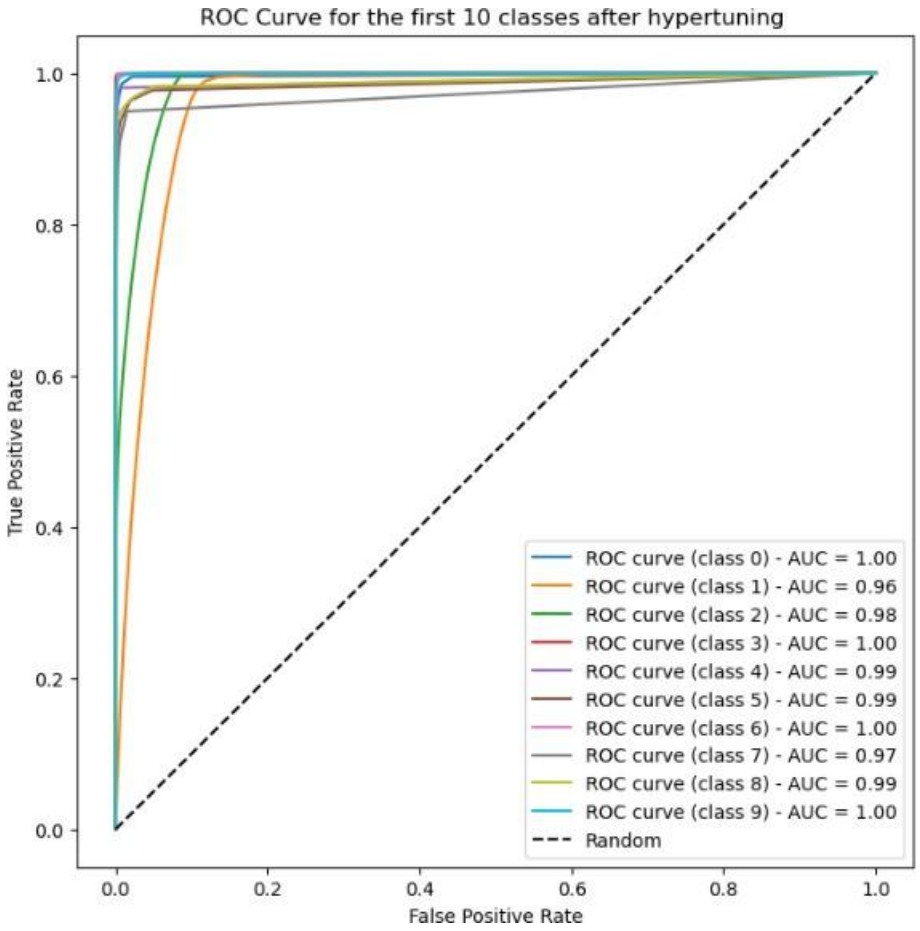
Confusion matrix of Testing dataset of Hyper parameter tuned model



Note. Confusion matrix of Testing dataset of Hyper parameter tuned model.

Figure 67

ROC plot of Hyperparameter tuned model



Note. ROC plot of Hyperparameter tuned model.

XGBoost:

Baseline Model. The XGBoost baseline model was configured with specific parameters: 'n_estimators' set to 75 trees, 'max_depth' set to 6 levels, 'learning_rate' of 0.1, utilizing 'mlogloss' as the evaluation metric, and the 'objective' defined as 'multi:softmax'. The dataset underwent a stratified split into training (70%), validation (20%), and test sets (10%). The training phase completed in around 5.86 seconds. Upon evaluation, the model exhibited an F1 score of 0.90 and a loss of 0.23, maintaining consistent performance across both validation and

training sets, achieving an accuracy of 0.89. Notably, these evaluations were conducted on imbalanced data.

Figure 68 and Figure 69 shows the model's classification report across 35 distinct crime types, the model demonstrated an overall accuracy of 0.90 and the F1-Score, which indicates the balance between precision and recall, was reported at 0.89. The F1-Score is crucial in assessing the model's ability to correctly identify instances of each class, particularly beneficial in these scenarios where classes are imbalanced.

Figure 68

Classification report of Test Dataset

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.58	0.68	1324
1	0.60	0.41	0.49	51398
2	0.79	0.90	0.85	143511
3	0.98	0.98	0.98	42545
4	0.94	0.37	0.53	123
5	0.75	0.79	0.77	2711
6	0.95	0.99	0.97	89245
7	0.69	0.56	0.62	716
8	0.97	0.93	0.95	21278
9	0.92	0.94	0.93	34945
11	0.92	0.67	0.77	1445
12	0.85	0.81	0.83	1271
13	0.00	0.00	0.00	7
14	0.85	0.34	0.48	1784
15	0.51	0.17	0.26	523
16	0.63	0.29	0.40	712
17	0.85	0.61	0.71	1496
18	0.97	0.92	0.95	38865
19	0.99	0.99	0.99	74287
20	0.00	0.00	0.00	3
21	1.00	0.04	0.07	26
23	0.00	0.00	0.00	82
24	0.84	0.72	0.77	5605
25	0.00	0.00	0.00	13
26	0.93	0.94	0.94	48680
27	0.94	0.77	0.84	7011
28	0.00	0.00	0.00	15
29	0.78	0.68	0.73	5312
30	0.00	0.00	0.00	2
31	0.93	0.97	0.95	29352
32	0.67	0.59	0.62	3159
33	0.00	0.00	0.00	477
34	0.98	0.99	0.99	164903
35	0.95	0.95	0.95	10910
accuracy			0.91	783736
macro avg	0.68	0.56	0.59	783736
weighted avg	0.90	0.91	0.90	783736

Note. Classification report of Test Dataset.

Figure 69

Classification report of Validation Dataset

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.59	0.69	2683
1	0.61	0.41	0.49	104630
2	0.79	0.90	0.84	290171
3	0.98	0.98	0.98	86378
4	0.95	0.32	0.48	238
5	0.76	0.78	0.77	5696
6	0.95	0.99	0.97	181609
7	0.71	0.56	0.63	1579
8	0.97	0.93	0.95	43429
9	0.92	0.94	0.93	70841
11	0.90	0.67	0.77	2915
12	0.86	0.84	0.85	2583
13	0.00	0.00	0.00	25
14	0.82	0.31	0.45	3696
15	0.51	0.23	0.31	935
16	0.65	0.29	0.41	1462
17	0.85	0.61	0.71	3001
18	0.97	0.92	0.95	78306
19	0.99	0.99	0.99	150383
20	0.00	0.00	0.00	7
21	0.67	0.04	0.08	45
23	0.50	0.01	0.01	171
24	0.85	0.71	0.77	11375
25	0.00	0.00	0.00	29
26	0.93	0.94	0.94	98843
27	0.94	0.77	0.85	14240
28	0.00	0.00	0.00	38
29	0.77	0.67	0.72	10502
30	0.00	0.00	0.00	4
31	0.93	0.97	0.95	59699
32	0.66	0.58	0.62	6429
33	0.00	0.00	0.00	1048
34	0.98	0.99	0.99	335975
35	0.95	0.95	0.95	22254
accuracy			0.91	1591219
macro avg	0.68	0.56	0.59	1591219
weighted avg	0.90	0.91	0.90	1591219

Note. Classification report of Validation Dataset.

The ROC curve presents the performance of the machine learning algorithm. It shows the trade-off between sensitivity and specificity. Sensitivity is the ability of the model to identify positive cases, while specificity is the ability of the model to identify negative cases. Figure 70 shows ROC curve before tuning the data. And it shows that the model is performing well, with an AUC of 1.0 for most classes. This means that the model is very good at distinguishing between positive and negative cases. Figure 71 and Figure 72 below displays the confusion matrix, illustrating the count of misclassifications for each class, evaluating the model's performance on both the validation and test sets.

Figure 70

ROC curve of XGBoost

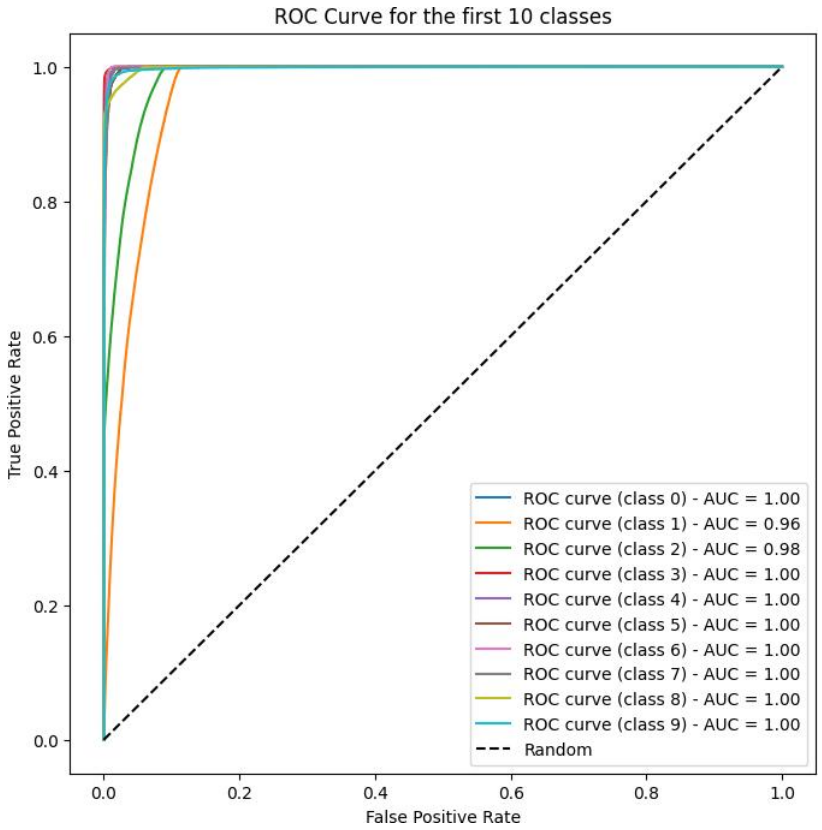
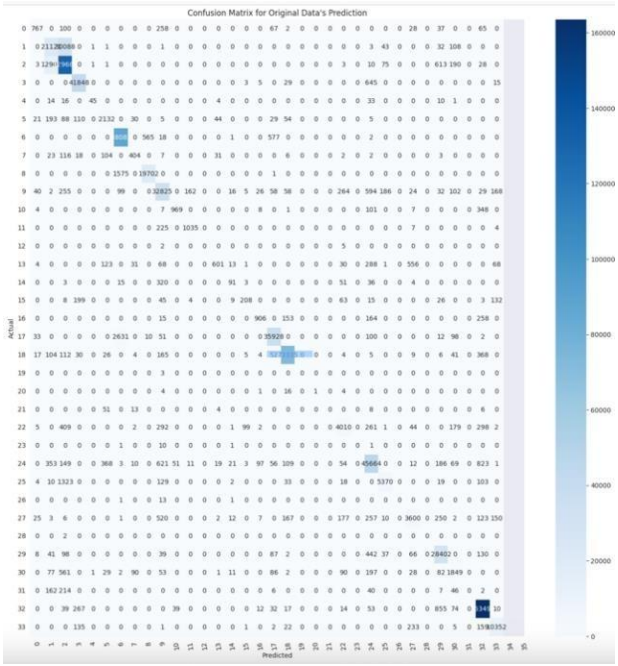


Figure 71

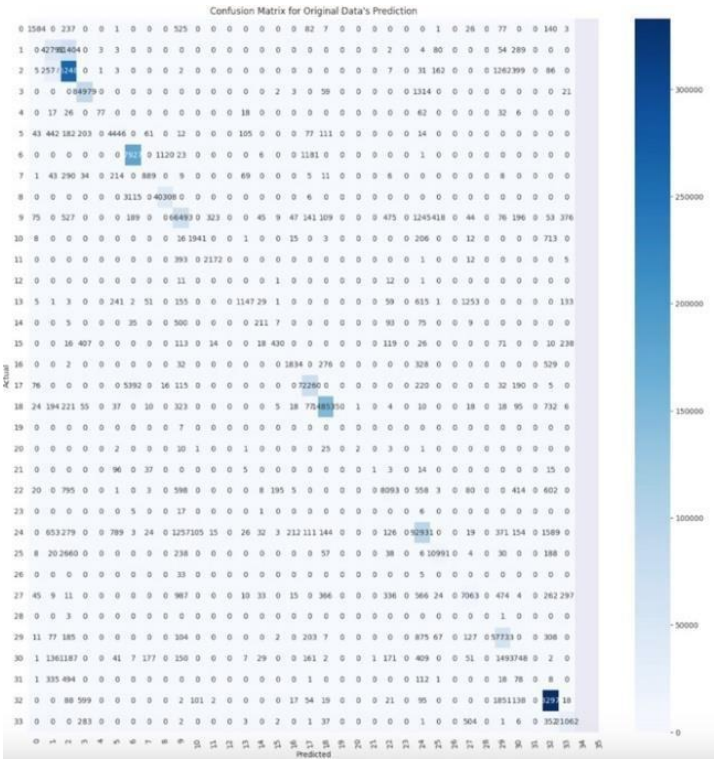
Confusion matrix of Test Dataset



Note. Confusion matrix for test dataset.

Figure 72

Confusion matrix of Validation Dataset



Note. Confusion matrix for validation dataset.

Hyperparameter tuned model. The baseline model initially achieved an F1-Score of 0.90, which led to an exploration of hyperparameter tuning. Utilizing GridSearchCV, the combinations of 'n_estimators' (100, 125) and 'learning_rate' (0.01) were optimized. Post the tuning process, the identified best parameters were 'n_estimators' set at 125 and 'learning_rate' at 0.01. However, despite investing more time in training to optimize these hyperparameters, the model achieved an F-Score of 0.89 on the test data. This suggests that the specific tuning of 'n_estimators' and 'learning_rate' at 0.01 did not result in an improved accuracy, indicating that the model's performance might have already been near an optimal value. Tuning a particular hyperparameter in this manner did not significantly impact the model's performance. Figure 73

and figure 74 shows the best parameters which are obtained and the classification report and confusion matrix that of test data set.

Figure 73

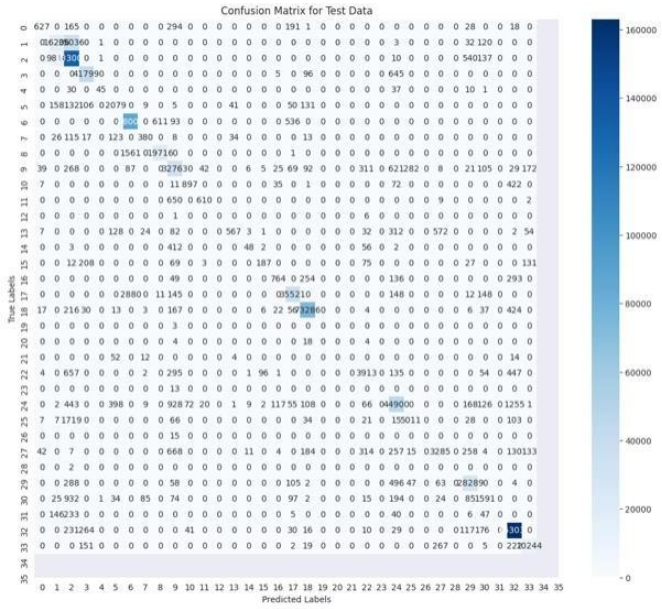
Classification Report for Hyperparameter tuned model

```
Best Parameters: {'learning_rate': 0.01, 'n_estimators': 100}
Validation Accuracy with Best Model: 90.15%
Test Accuracy with Best Model: 90.18%
              precision    recall  f1-score   support

           0       0.84      0.47      0.60      1324
           1       0.61      0.32      0.42     51398
           2       0.77      0.93      0.84    143511
           3       0.98      0.98      0.98     42545
           4       0.94      0.37      0.53       123
           5       0.74      0.77      0.75      2711
           6       0.95      0.99      0.97     89245
           7       0.73      0.53      0.61       716
           8       0.97      0.93      0.95     21278
           9       0.89      0.94      0.91     34945
          11       0.89      0.62      0.73      1445
          12       0.90      0.48      0.63      1271
          13       0.00      0.00      0.00         7
          14       0.88      0.32      0.47      1784
          15       0.62      0.09      0.16       523
          16       0.63      0.26      0.37       712
          17       0.79      0.51      0.62      1496
          18       0.97      0.91      0.94     38865
          19       0.99      0.99      0.99     74287
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00        26
          23       0.00      0.00      0.00        82
          24       0.81      0.70      0.75      5605
          25       0.00      0.00      0.00        13
          26       0.93      0.92      0.93     48680
          27       0.94      0.71      0.81      7011
          28       0.00      0.00      0.00        15
          29       0.78      0.62      0.69      5312
          30       0.00      0.00      0.00         2
          31       0.92      0.96      0.94     29352
          32       0.65      0.50      0.57      3159
          33       0.00      0.00      0.00       477
          34       0.98      0.99      0.98    164903
          35       0.95      0.94      0.95     10910

    accuracy                           0.90    783736
   macro avg       0.65      0.52      0.56    783736
weighted avg       0.90      0.90      0.89    783736
```

Note. Classification Report for Hyperparameter tuned model.

Figure 74

Confusion matrix for Hyperparameter tuned model

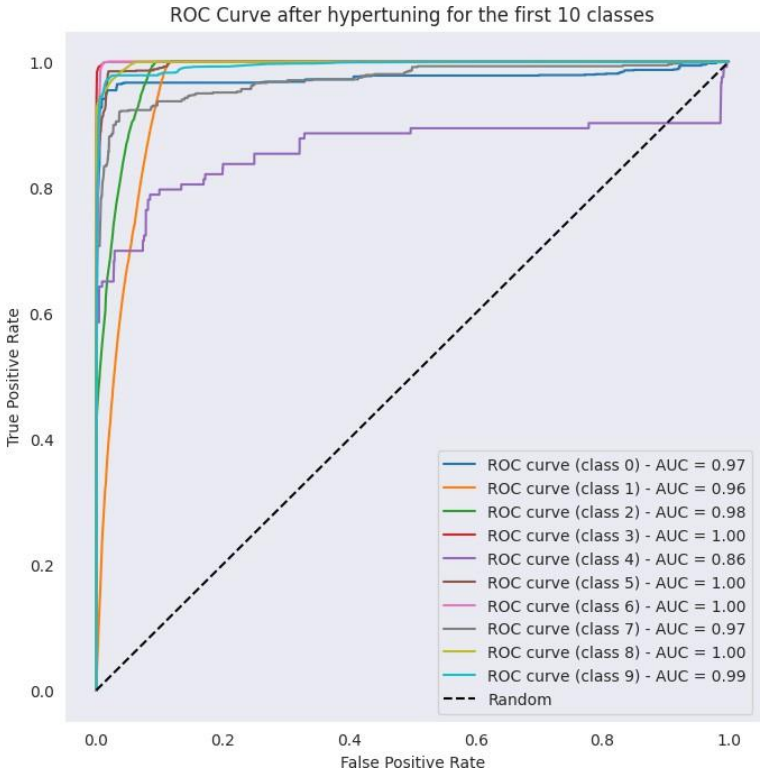


Note. Confusion matrix for Hyperparameter tuned model.

Figure 75 shows ROC curve after hyper parameter tuning on test data. The AUC scores for most of the classes are above 0.9, which indicates that the model is able to distinguish between positive and negative cases with a high degree of accuracy and the ROC curve is not symmetrical for some of the classes. This indicates that the model is more likely to make false positives or false negatives for certain classes than for others.

Figure 75

AUC-ROC curve for Hyperparameter tuned model



Note. AUC-ROC curve for Hyperparameter tuned model.

Testing Data. The hyper-tuned model underwent evaluation on a previously unseen 10% of the data, resulting in an F-score of 0.89, and a log loss of 0.19. A comprehensive evaluation of this model encompassed various metrics, including f1-score, recall, and precision, as depicted in figure_. These metrics were observed both before and after tuning, highlighting the model's comprehensive predictive capacity across diverse classes. This signifies its reliability in accurately distinguishing and categorizing different instances within the dataset. The comparison, outlined in figure 76 and figure 77, illustrates the classification report for the test data before and after the tuning process. After tuning, the model's performance on the test data exhibited a slight decrease in F-Score by 1% and overall, it is an optimal value.

Figure 76

Before Hyperparameter Tuning for test data

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.58	0.68	1324
1	0.60	0.41	0.49	51398
2	0.79	0.90	0.85	143511
3	0.98	0.98	0.98	42545
4	0.94	0.37	0.53	123
5	0.75	0.79	0.77	2711
6	0.95	0.99	0.97	89245
7	0.69	0.56	0.62	716
8	0.97	0.93	0.95	21278
9	0.92	0.94	0.93	34945
11	0.92	0.67	0.77	1445
12	0.85	0.81	0.83	1271
13	0.00	0.00	0.00	7
14	0.85	0.34	0.48	1784
15	0.51	0.17	0.26	523
16	0.63	0.29	0.40	712
17	0.85	0.61	0.71	1496
18	0.97	0.92	0.95	38865
19	0.99	0.99	0.99	74287
20	0.00	0.00	0.00	3
21	1.00	0.04	0.07	26
23	0.00	0.00	0.00	82
24	0.84	0.72	0.77	5605
25	0.00	0.00	0.00	13
26	0.93	0.94	0.94	48680
27	0.94	0.77	0.84	7011
28	0.00	0.00	0.00	15
29	0.78	0.68	0.73	5312
30	0.00	0.00	0.00	2
31	0.93	0.97	0.95	29352
32	0.67	0.59	0.62	3159
33	0.00	0.00	0.00	477
34	0.98	0.99	0.99	164903
35	0.95	0.95	0.95	10910
accuracy			0.91	783736
macro avg	0.68	0.56	0.59	783736
weighted avg	0.90	0.91	0.90	783736

Note. Before Hyperparameter Tuning Classification Report for test data.

Figure 77

After Hyperparameter Tuning for test data

```
Best Parameters: {'learning_rate': 0.01, 'n_estimators': 100}
Validation Accuracy with Best Model: 90.15%
Test Accuracy with Best Model: 90.18%
              precision    recall  f1-score   support

           0       0.84      0.47      0.60      1324
           1       0.61      0.32      0.42     51398
           2       0.77      0.93      0.84    143511
           3       0.98      0.98      0.98     42545
           4       0.94      0.37      0.53       123
           5       0.74      0.77      0.75      2711
           6       0.95      0.99      0.97     89245
           7       0.73      0.53      0.61       716
           8       0.97      0.93      0.95     21278
           9       0.89      0.94      0.91     34945
          11       0.89      0.62      0.73      1445
          12       0.90      0.48      0.63      1271
          13       0.00      0.00      0.00         7
          14       0.88      0.32      0.47      1784
          15       0.62      0.09      0.16       523
          16       0.63      0.26      0.37       712
          17       0.79      0.51      0.62      1496
          18       0.97      0.91      0.94     38865
          19       0.99      0.99      0.99     74287
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00        26
          23       0.00      0.00      0.00        82
          24       0.81      0.70      0.75      5605
          25       0.00      0.00      0.00        13
          26       0.93      0.92      0.93     48680
          27       0.94      0.71      0.81      7011
          28       0.00      0.00      0.00        15
          29       0.78      0.62      0.69      5312
          30       0.00      0.00      0.00         2
          31       0.92      0.96      0.94     29352
          32       0.65      0.50      0.57      3159
          33       0.00      0.00      0.00       477
          34       0.98      0.99      0.98    164903
          35       0.95      0.94      0.95     10910

    accuracy                           0.90    783736
   macro avg       0.65      0.52      0.56    783736
weighted avg       0.90      0.90      0.89    783736
```

Note. After Hyperparameter Tuning Classification Report for test data.

Artificial Neural Network:

Baseline Model. The baseline ANN model is implemented using TensorFlow's Keras module. Only one hidden layer with 64 neurons is added to this neural network, the activation function is decided as 'ReLu'. The softmax function is used in the output layer to convert the results into probabilities for each class. Epoch (number of iterations over the entire dataset) is set to 5; batch_size (number of instances processed before updating the weights) is set to 32; cross entropy is selected as the loss function to evaluate the model. It's trained on the original imbalanced training dataset, which includes 5,548,500 records of crimes with 6 features and 36 different labels. The model performance is evaluated during the training process after each epoch using the dataset specified by the 'validation_data' parameter. Figure 78 shows the loss, and accuracy on the training and validation dataset, and it takes 130s for each epoch to train. After 5 epochs, the model gets a log loss of 0.58 and an F1 score of 0.80 on the training dataset.

Figure 78

ANN Baseline Model Training Report

```
Epoch 1/5
173391/173391 [==============================] - 131s 752us/step - loss: 1.0410 - accuracy: 0.6813 - val_loss: 0.83
01 - val_accuracy: 0.7327
Epoch 2/5
173391/173391 [==============================] - 134s 774us/step - loss: 0.7256 - accuracy: 0.7638 - val_loss: 0.66
53 - val_accuracy: 0.7861
Epoch 3/5
173391/173391 [==============================] - 132s 763us/step - loss: 0.6398 - accuracy: 0.7886 - val_loss: 0.61
90 - val_accuracy: 0.7938
Epoch 4/5
173391/173391 [==============================] - 130s 748us/step - loss: 0.6031 - accuracy: 0.7973 - val_loss: 0.59
33 - val_accuracy: 0.7925
Epoch 5/5
173391/173391 [==============================] - 132s 763us/step - loss: 0.5795 - accuracy: 0.8035 - val_loss: 0.57
21 - val_accuracy: 0.8066
```

Note. ANN Baseline Model Training Report.

Hyperparameter Tuned Model. The baseline accuracy is good. However, it is always worthwhile to do a grid search to see if we can get a higher F1 score. After testing, we find the optimal hyperparameters as the number of hidden layers equals five, 'epochs'=10, 'batch_size'=64, and the others remain the same. The performance of the hyper tuned model is shown as Figure 79. We finally achieved a loss of 0.19 and an accuracy of 0.92. Additionally, the running time for each epoch reduced from 130s to 90s as we adjusted batch size from 32 to 64.

Figure 79

ANN Hyper Parameter Tuned Model Training Report

```
Epoch 1/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.4676 - accuracy: 0.8262 - val_loss: 0.4179 -
val_accuracy: 0.8565
Epoch 2/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.3078 - accuracy: 0.8746 - val_loss: 0.3175 -
val_accuracy: 0.8746
Epoch 3/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.2589 - accuracy: 0.8922 - val_loss: 0.2505 -
val_accuracy: 0.8914
Epoch 4/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.2330 - accuracy: 0.9017 - val_loss: 0.2526 -
val_accuracy: 0.8940
Epoch 5/10
86696/86696 [==============================] - 87s 1ms/step - loss: 0.2147 - accuracy: 0.9084 - val_loss: 0.2307 -
val_accuracy: 0.8983
Epoch 6/10
86696/86696 [==============================] - 88s 1ms/step - loss: 0.2040 - accuracy: 0.9119 - val_loss: 0.1914 -
val_accuracy: 0.9144
Epoch 7/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.1961 - accuracy: 0.9146 - val_loss: 0.1707 -
val_accuracy: 0.9242
Epoch 8/10
86696/86696 [==============================] - 729s 8ms/step - loss: 0.1921 - accuracy: 0.9160 - val_loss: 0.1795 -
val_accuracy: 0.9180
Epoch 9/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.1891 - accuracy: 0.9171 - val_loss: 0.2326 -
val_accuracy: 0.8992
Epoch 10/10
86696/86696 [==============================] - 89s 1ms/step - loss: 0.1864 - accuracy: 0.9180 - val_loss: 0.1780 -
val_accuracy: 0.9209
```

Note. ANN Hyper Parameter Tuned Model Training Report.

Test Data. Figure 80 shows the average accuray, precision, recall, and f1-score in addition to each class. We've got a f1-score of 0.79, matching the training performance. The macro avg is relatively lower as the data is imbalanced.

Figure 80

Classification Report For Baseline Model

	precision	recall	f1-score	support
0	0.78	0.37	0.50	1281
1	0.48	0.31	0.37	51657
2	0.74	0.87	0.80	143368
3	0.89	0.90	0.90	42516
4	0.47	0.26	0.34	103
5	0.77	0.52	0.62	2775
6	0.82	0.96	0.88	89570
7	0.61	0.52	0.56	752
8	0.82	0.20	0.32	21757
9	0.61	0.55	0.58	34693
11	0.75	0.55	0.64	1504
12	0.41	0.51	0.45	1255
13	0.00	0.00	0.00	10
14	0.71	0.33	0.46	1860
15	0.00	0.00	0.00	451
16	0.33	0.00	0.01	719
17	0.43	0.14	0.21	1514
18	0.77	0.90	0.83	38549
19	0.96	0.90	0.93	74439
20	0.00	0.00	0.00	3
21	0.00	0.00	0.00	15
22	0.00	0.00	0.00	1
23	0.00	0.00	0.00	67
24	0.74	0.28	0.40	5654
25	0.00	0.00	0.00	16
26	0.75	0.84	0.79	48629
27	0.71	0.81	0.76	6908
28	0.00	0.00	0.00	20
29	0.53	0.48	0.51	5217
30	0.00	0.00	0.00	3
31	0.83	0.69	0.75	29495
32	0.40	0.01	0.02	3131
33	0.00	0.00	0.00	503
34	0.92	0.94	0.93	165336
35	0.74	0.93	0.82	10946
accuracy			0.81	784717
macro avg	0.49	0.39	0.41	784717
weighted avg	0.80	0.81	0.79	784717

Note. Classification Report For Baseline Model.

Figure 81

Figure 81 shows the hyper-tuned model performance. The model obtains an f1-score of 0.91, when it is evaluated with 10% test data prepared in the data engineering chapter. Figure 82 and Figure 83 displays the confusion matrix for the hyper tuned model, demonstrating the classification scenarios for each crime type.

Figure 82

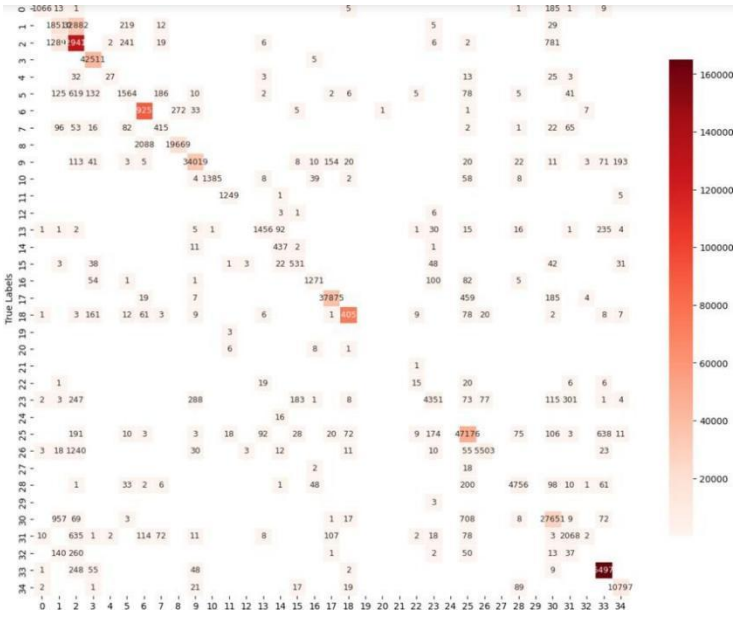
Classification Report for Hyper Tuned Model

	precision	recall	f1-score	support
0	0.98	0.83	0.90	1281
1	0.56	0.36	0.44	51657
2	0.78	0.90	0.84	143368
3	0.99	1.00	0.99	42516
4	0.87	0.26	0.40	103
5	0.72	0.56	0.63	2775
6	0.97	1.00	0.99	89570
7	0.58	0.55	0.57	752
8	0.99	0.90	0.94	21757
9	0.99	0.98	0.98	34693
11	1.00	0.92	0.96	1504
12	0.98	1.00	0.99	1255
13	0.00	0.00	0.00	10
14	0.91	0.78	0.84	1860
15	0.75	0.97	0.84	451
16	0.69	0.74	0.71	719
17	0.92	0.84	0.88	1514
18	0.99	0.98	0.99	38549
19	1.00	0.99	1.00	74439
20	0.00	0.00	0.00	3
21	0.00	0.00	0.00	15
22	0.00	0.00	0.00	1
23	0.36	0.22	0.28	67
24	0.92	0.77	0.84	5654
25	0.00	0.00	0.00	16
26	0.96	0.97	0.96	48629
27	0.98	0.80	0.88	6908
28	0.00	0.00	0.00	20
29	0.95	0.91	0.93	5217
30	0.00	0.00	0.00	3
31	0.94	0.94	0.94	29495
32	0.81	0.66	0.73	3131
33	0.00	0.00	0.00	503
34	0.99	1.00	1.00	165336
35	0.98	0.99	0.98	10946
accuracy			0.92	784717
macro avg	0.67	0.62	0.64	784717
weighted avg	0.91	0.92	0.91	784717

Note. Classification Report for Hyper Tuned Model.

Figure 83

Confusion Matrix for Hyper Tuned Model

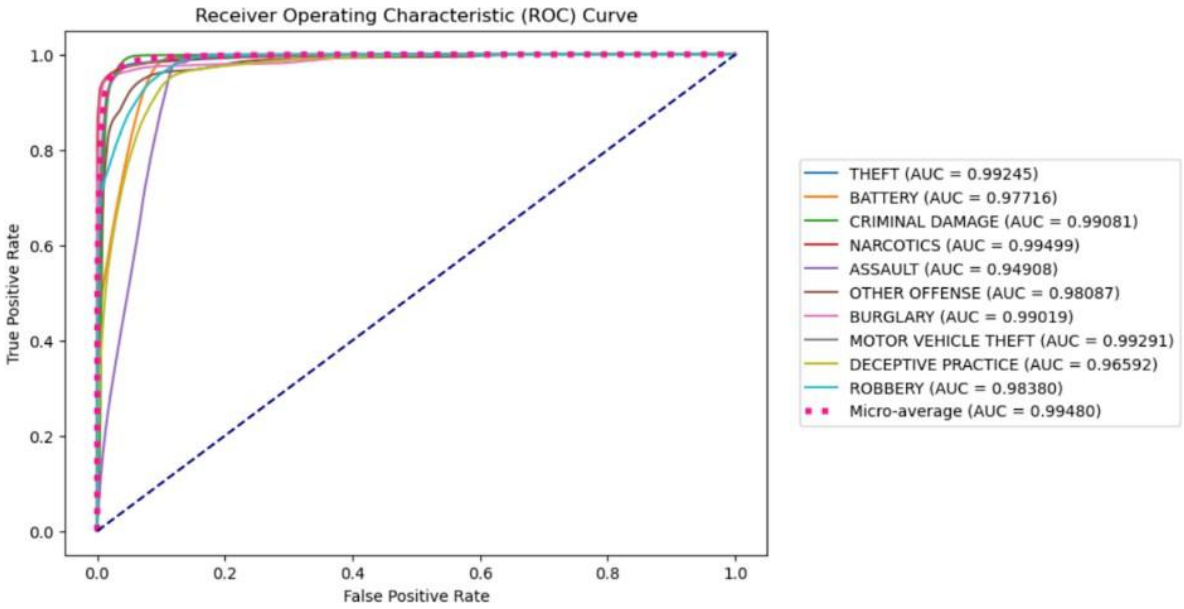


Note. Confusion Matrix for Hyper Tuned Model.

Figures 84 and 85 display the ROC-AUC for the baseline and hyper-tuned models, respectively. Despite a notable difference in accuracy between these two models, the AUC scores are similar, both micro-averages reaching 0.99. This observation suggests that AUC scores can be optimistic when it's handling imbalanced data. In our crime prediction problem, the frequency of each crime type varies from 1.6e to 10.

Figure 84

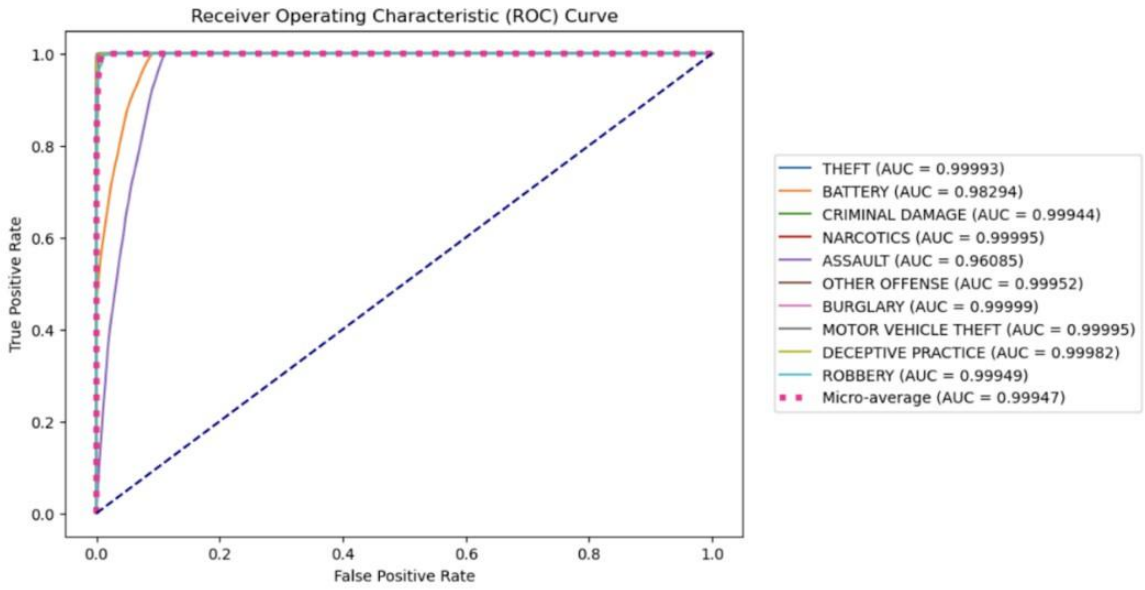
The ROC for Baseline Model



Note. The ROC for Baseline Model.

Figure 85

The ROC for Hypertuned Model



Note. The ROC for Hypertuned Model.

(Anonymous) Sharing agreement:

- Do you agree to share your work as an example for next semester? - Yes
- Do you want to hide your name/team if you agree? - Yes

References

Abdulraheem, M., Awotunde, J. B., Oladipo, I. D., Adeleke, M. O., Ndunagu, J. N., Ayantola, J. A., & Mohammed, A. (2022). *Crime rate prediction using the random forest algorithm* Retrieved October 23, 2023, from https://core.ac.uk/reader/560325700

Ahishakiye, E., Taremwa, D., Omulo, E., Niyonzima, I. (2017). *Crime prediction using decision tree (j48) classification algorithm.* Retrieved October 23, 2023, from https://www.ijcit.com/archives/volume6/issue3/Paper060308.pdf

Aldossari, N., Algefes, A., Masmoudi, F., & Kariri, E. (2022). *Data science approach for crime analysis and prediction*: Saudi Arabia use-case. Retrieved October 23, 2023, from https://ieeexplore.ieee.org/document/9764853

https://link-springer-com.libaccess.sjlibrary.org/content/pdf/10.1007/s10666-018-9609-3

Bharati, A., Dr Sarvanaguru, RA.K. (2018). *Crime prediction and analysis using machine learning.* Retrieved October 23, 2023, from https://www.irjet.net/archives/V5/i9/IRJET-V5I9192.pdf

Chen, T., & Guestrin, C. (2016). *XGBoost: a scalable tree boosting system.* Retrieved October 23, 2023, from https://arxiv.org/pdf/1603.02754.pdf

Cherif, I. L., & Kortebi, A. (2019). *On using extreme gradient boosting (xgboost) machine learning algorithm for home network traffic classification.* Retrieved October 23, 2023, from https://ieeexplore-ieee-org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=8734193

Dongare, A. D., Kharde, R. R., & Kachare, A. D. (2012*). Introduction to artificial*

neural network. Retrieved October 23, 2023, from https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04d0b6952a4f0c7203577afc9476c2fcab2cba06

Guo, H., Nguyen, H., Vu, D., Bui, X. (2021). *Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach.* Retrieved October 23, 2023, from https://www-sciencedirect-com.libaccess.sjlibrary.org/science/article/pii/S0301420718306901?via%3Dihub

Hermawan, F., Prianggono, J. (2023). *Crime of theft prediction using machine learning K-Nearest neighbour algorithm at polresta bandar lampung.* Retrieved October 23, 2023, from https://jurnal.polgan.ac.id/index.php/sinkron/article/view/12422/1751

Kumar, A., Verma, A., Shinde, G., Sukhdeve, Y. (2020). *Crime prediction using k-nearest neighboring algorithm*. Retrieved October 23, 2023, from https://www.researchgate.net/publication/340969457_Crime_Prediction_Using_K-Nearest_Neighboring_Algorithm

Kwon, E., Jung, S., & Lee, J. (2021). *Artificial neural network model development to predict theft types in consideration of environmental factors.* Retrieved October 23, 2023, from https://www.mdpi.com/2220-9964/10/2/99

Lin, Y. L., Chen, T. Y., and Yu, L. C. (2017). *Using machine learning to assist crime prevention*. Retrieved October 23, 2023, from https://ieeexplore.ieee.org/abstract/document/8113405

Walczak S. (2021). *Predicting crime and other uses of neural networks in police decision making.* Retrieved October 23, 2023, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8529125/