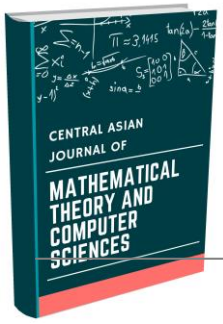

CENTRAL ASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES

https://cajmtcs.centralasianstudies.org

Volume: 04 Issue: 12 | Dec 2023 ISSN: 2660-5309

Simulating and Modeling a Secure Content Caching System in IoT Networks Using Deep Learning Tools

Ahmed Adnan Hadi

Open Educational College, Al-Qadisiyah Center, Iraq

Abstract

By keeping frequently accessed content closer to the end users, content caching is essential to maximizing the performance of Internet of Things (IoT) networks. Nonetheless, because the data being transferred is sensitive, security in IoT situations is critical. The goal of this project is to use deep learning methods, namely decision trees and neural networks, to simulate and build a secure content caching system in Internet of Things networks.

The suggested method makes use of decision trees to identify the best material to cache according to a number of variables, including network conditions, security needs, and content popularity. Because decision trees recursively divide data according to qualities, they are useful for managing complex scenarios and allow for precise caching decisions.

Moreover, neural networks are used to improve the caching system's security feature. The system is able to make security-aware decisions on content caching by using a neural network model that has been trained on a large dataset that includes different network settings and content attributes. By identifying patterns and connections between various input parameters, the neural network model is able to recognize possible security threats and modify the caching approach accordingly.

The caching hit ratio, latency reduction, and security enhancement can be used to assess the efficacy and performance of the secure content caching system through modeling and simulation. The study's findings will shed important light on the possible advantages of utilizing deep learning techniques to secure content caching systems for Internet of Things networks.

The overarching goal of this research is to improve the user experience and protect sensitive IoT data by aiding in the development of dependable and secure content caching technologies that increase the effectiveness and security of IoT networks.

ARTICLEINFO

Article history:
Received 13 Oct 2023
Revised form 20 Nov 2023
Accepted 22 Dec 2023

Keywords: *Secure content caching system - IoT networks - Deep learning tools – Simulating – Modeling - Content caching - Security in IoT - Machine learning algorithms*

---***---

Introduction

IoT and connected networks are examples of cutting-edge technology that allow items and gadgets to communicate and connect with one another online. With the use of this technology, we can now connect the digital and physical realms, creating new opportunities for clever and creative applications in a range of industries, including manufacturing, transportation, healthcare, and agriculture.

The prevalence and growing use of linked devices has made security concerns more crucial. The vast amount of data being transferred and the variety of devices and communications that are a part of IoT networks provide a number of security challenges. Among the security issues are:

- Phishing attacks: False messages are sent to users with the intention of stealing their login credentials or personal information.
- Distributed Denial of Service (DDoS) Attacks: These attacks cause service disruptions by artificially raising the volume of traffic on the network in order to target its resources.

Intrusion Attacks: When IoT device vulnerabilities are taken advantage of, unauthorized access to the network can be obtained, potentially resulting in data theft or system disruption.

This is where content storage systems come into play for improving IoT network performance and security. Reducing the amount of queries users send to central servers can be accomplished with the help of caching technologies. The storage system caches the content on devices that users interact with directly, saving users from constantly contacting distant servers to retrieve content. As a result, traffic volume is decreased and network performance is enhanced [1].

In terms of Artificial Intelligence (AI), Deep Learning is a subfield that focuses on teaching computer programs and models to recognize intricate patterns, communicate with one another, and handle data in a manner that is similar to that of a person. It is especially advantageous to use deep learning algorithms in content storage systems in Internet of Things networks. Deep learning can assist with content prediction, user activity analysis, and the selection of appropriate storage devices. In order to protect stored material, encryption protocols can also be applied with it.

Deep learning methods can be used to simulate and model a secure content storage system in IoT networks. This allows you to assess the system's security and performance in a range of real-world-like scenarios and make necessary improvements [2].

Literary study:

Researchers Shanthamallu, U. S., Spanias, A., Tepedelenlioglu, C., & Stanley, M. provide a brief overview of the fundamental ideas and algorithms used in machine learning and its applications, beginning with a more general definition. The study also covers a few sensor and Internet of Things applications of machine learning techniques. Next, we present a variety of learning techniques, such as deep learning models and supervised and unsupervised techniques. We also go over the use of machine learning algorithms in a number of domains, such as sensor networks, pattern identification, anomaly detection, the Internet of Things (IoT), and health monitoring. A few software tools were mentioned at the conclusion of the study [3].

The book "IoT enabled technology in secured healthcare: Applications, challenges and future directions" by Goyal, Sharma, Bhushan, Shankar, & Sagayam, M. provides a thorough analysis of the security issues that IoT networks face as well as information on how content caching might help.

The following applications of deep learning to enhance content caching in Internet of Things networks are covered in the book:

Finding the most popular content to cache: Deep learning can be used to examine past data in order to determine which information is most popular to cache. By lowering network traffic, this can aid in enhancing the caching system's performance.

Choosing the best devices to cache content: To choose the best devices to cache content, deep learning can be used to examine an IoT network's device characteristics. This can lower power usage and infrastructure expenses, which can increase the caching system's efficiency.

Putting safe encryption methods into practice: New, more attack-resistant encryption protocols can be created using deep learning. By doing this, you may be able to prevent unwanted access to the content.

Content caching solutions in Internet of Things networks could perform much better and be much more secure with the use of deep learning. Nevertheless, before deep learning is extensively used for this purpose, a few issues still need to be resolved.

Deep learning models require big datasets to be trained, which presents a hurdle. IoT networks may find this difficult because they frequently produce a lot of data. The requirement for specialized hardware to execute deep learning models presents another difficulty. For small and medium-sized firms, this could be a hurdle.

Deep learning is a promising technology that has the potential to completely change how content caching is employed in Internet of Things networks, despite these obstacles [4].

Al-Garadi, M. A., Al-Ali, A. K., Mohamed, A., Du, Integrating the Internet of Things (IoT) into billions of intelligent devices that can converse with one another with little assistance from humans is the goal of IoT security. One of the areas of computing that is developing the fastest right now is the Internet of Things, but new security concerns have emerged because of the comprehensive nature of IoT systems and the interdisciplinary components involved in their deployment. Implementing security mechanisms like encryption, authentication, access control, network security, and application security is pointless when it comes to Internet of Things (IoT) devices and their inherent vulnerabilities. The challenge has been solved by this research since ML/DL techniques are crucial in changing the security of IoT systems from just enabling secure device-to-device communication to security-based intelligence systems. This research is important because it offers a thorough overview of machine learning (ML) techniques and the latest advancements in deep learning (DL) techniques, which can be leveraged to create more secure Internet of Things systems. The many possible attack surfaces of the IoT system and the possible threats associated with each surface are explored, along with the IoT security threats related to latent or recently developed threats. After that, we go over ML/DL methods to IoT security in detail, outlining the benefits, drawbacks, and potential associated with each strategy. The study also covered the potential applications and difficulties of ML/DL for IoT security. Future research directions may be suggested by these opportunities and challenges [5].

A number of researchers have been interested in the field of intelligent transportation, which has been explored through the application of Internet of Things and machine learning techniques. Zantalis, F., Koulouras, G., Karabetsos, S., & Kandris, D. examine machine learning and IoT in smart transportation, defining smart transportation as a catch-all for parking, street lights, route optimization, detection of anomalies on the road, and infrastructure applications. In order to have a comprehensive understanding of trends in the aforementioned fields and to identify prospective coverage needs, the research conducted an independent review of Internet of Things applications and machine learning methods in intelligent transportation systems (ITS)The study also made clear that machine learning may not be sufficiently covered for applications such as smart parking and smart lighting systems. Furthermore, researchers find that parking, accident/detection, and route optimization are the most often used ITS applications [6].

Methodology:

To simulate and design a secure content caching system in IoT networks using deep learning tools, you can follow these general steps:

- **Data Collection**

We gather a sample dataset comprising security attributes, content properties, and network metrics. Make sure the dataset is representative of real-world IoT network settings and encompasses a wide range of

scenarios. The deep learning model will be trained and assessed using this dataset. Preparing the gathered data set for deep learning model training is known as data preprocessing. This stage could involve feature extraction, categorical variable coding, and data cleaning and normalization [7]. Making sure the data accurately represents features and labels and is in the right format is crucial. Select a deep learning architecture that is suitable for your content caching system. This could be a recurrent neural network (RNN), a convolutional neural network (CNN), or a hybrid of the two. Consider the particular needs of your system while creating the form's structure, as well as the type of data that will be entered. Create training and validation sets from your data source. Utilizing the training set, train a deep learning model. The model learns how to relate intended outputs (security attributes) to input features (network parameters, content qualities) during training [8]. In order to reduce the model loss function, gradient optimization and forward and reverse propagation are used in this process. Utilizing a validation set, assess the trained model's performance. Determine pertinent metrics to assess the model's predictive accuracy of security features, including precision, recall, F1 score, and precision. Run the test using a different test dataset after evaluating the model to see how well it performs with unknown data. Based on the findings of the evaluation, modify the model as needed to enhance its performance [10–9].

- **Data Preprocessing:**

Data preprocessing is essential for preparing the dataset for training and enhancing model performance when simulating and modeling a secure content caching system in IoT networks using deep learning technologies. You can use the following popular data preparation methods [11]:

1. Data cleaning: Take out of the gathered data set any errors, inconsistencies, or outliers. This can entail locating and resolving duplicates, noisy data, or missing values. Depending on the type of data, you can use methods like imputation to replace missing values or eliminate instances where there are none at all.
2. Normalization: To guarantee that the input features have comparable scales, normalize them to a common scale. When working with numerical properties that have several scales or units, this step is extremely crucial. Z-score normalization and minimum and maximum scaling are examples of common normalization methods.
3. Feature extraction: Take pertinent features out of the raw data that can give the content caching system useful insights. This could entail developing new features using feature engineering techniques or domain expertise, or it could entail aggregating or modifying already-existing features. From network parameters and content attributes, for instance, statistical features, time-based features, or frequency-based features can be extracted.
4. Dimensionality reduction: To minimize the number of features and eliminate superfluous or unnecessary data, take into account using dimensionality reduction approaches if the dataset has high-dimensional features. Principal component analysis (PCA) and distributed stochastic neighbor embedding (t-SNE) are two widely used dimensionality reduction techniques.
5. Encoding category variables: If there are categorical variables in the dataset, encode them so that deep learning models can use them as numerical representations. Depending on the needs of the model and the type of categorical variables, common coding strategies include label coding, imputation, and single coding. Even when the code is operating offline, the precise data pretreatment methods you use may change based on the features of your dataset and the specifications of your content-secure caching solution. Which preprocessing methods are most appropriate for your particular use case depends in large part on experimentation and domain expertise.

Decision tree methods can be used to simulate and build a secure content caching system in Internet of Things networks, even though they are not usually categorized as deep learning tools. Based on the characteristics and their significance in the preprocessed dataset, decision tree models are able to choose the best caching choices. Using decision tree modeling in your system can be done as follows [12]:

- **Decision Tree Modeling**:

1. Preparing the Dataset: Make sure your preprocessed dataset has the properties and labels required to train the decision tree model. The labels should show the best caching choices, and the attributes should reflect pertinent network metrics, content characteristics, and security features.
2. Dataset Splitting: Split your data set into a testing set and training set. The decision tree model will be constructed using the training set, and its performance will be assessed using the testing set.
3. Selection of Decision Tree method: To build the model, select an appropriate decision tree method. ID3, C4.5, Random Forest, and CART (Classification and Regression Trees) are a few well-known decision tree algorithms. Choose the algorithm that best meets your needs as each one has unique features and factors to take into account.
4. Decision Tree Model Training: Train the decision tree model with the training dataset. The method will split the data recursively based on values of attributes in order to generate decision nodes. Depending on the algorithm selected, the splitting criteria may be based on information gain, entropy, or the Gini index [13].
5. Model Evaluation: Using the testing dataset, assess the decision tree model's performance once it has been trained. Analyze pertinent measures including recall, accuracy, precision, and F1 score to evaluate the model's predictive power for the best caching choices.
6. Fine-tuning and Validation: To enhance the decision tree model's performance, make appropriate adjustments to its hyperparameters or pruning methods. To make sure the model is reliable and broadly applicable, validate it using a different validation dataset.
7. Caching Decision Making: In actual IoT network settings, you may use the decision tree model to make caching decisions after it has been trained and validated. Once you enter the pertinent network parameters and content characteristics into the model, it will assist you in selecting the best caching strategy based on the importance of the attributes and patterns you have learnt.

Recall that decision tree models have drawbacks of their own, including overfitting and sensitivity to minute alterations in the dataset. Selecting the right method is crucial, as is taking into account alternative modeling strategies, such as deep learning models [14].

- **Neural Network Training**:

Learning a neural network model through training can be a useful method for capturing intricate patterns and correlations between network parameters and content features in the simulation and modeling of a secure content caching system in IoT networks using deep learning techniques. An outline of training a neural network model is provided here [16–15]:

1. Design of Neural Network Architecture: Create a neural network design that works well with your system for secure content caching. Think about the type of input elements, the intricacy of the relationships you wish to record, and the intended result. For sequential data, for instance, you may use a recurrent neural network (RNN), a feedforward neural network, or a convolutional neural network (CNN) for content based on images. Ascertain the quantity of layers, the kind of activation functions, and the interlayer linkages.
2. Preparing the Dataset: Make sure your preprocessed dataset has the input features (content attributes, network parameters) and matching output labels (best caching choices, security features) needed to train the neural network model. Divide the dataset, in a manner akin to decision tree modeling, into training and testing sets.
3. Model Training: Use suitable deep learning methods, like backpropagation, to train the neural network model. By maximizing a loss function, the model learns during training how to translate the input

features to the desired output labels. In order to minimize the loss, backpropagation computes the gradient of the loss function with respect to the model's parameters and modifies the network's weights and biases. The model's parameters are iteratively changed during this process to enhance performance.

4. Hyperparameter tuning: To determine the ideal setup, play around with the neural network model's hyperparameters, such as batch size, learning rate, number of hidden units, and regularization strategies. Trial and error or methods like grid search or random search can be used to accomplish this.
5. Model Evaluation: Utilizing the testing dataset, assess the trained neural network model's performance. Examine pertinent measures like recall, accuracy, precision, and F1 score to evaluate the model's predictive power for the best caching choices or security features. Furthermore, you can use ROC curves, confusion matrices, and visualizations to assess the model's performance.
6. Iteration and fine-tuning: Adjust the neural network model as needed in light of the evaluation's findings. To further enhance its performance, modify the regularization strategies, hyperparameters, or architecture. Continue doing this until you get the desired outcomes.
7. Deployment and Inference: You can use a trained and verified neural network model in a simulated or real-world Internet of Things network. Make caching decisions using the model by taking into account the properties of the content and the input network parameters. The model will anticipate or suggest the best content caching strategies based on the relationships and patterns it has learned.

Result

To read data straight from an Excel file into a Data Frame, use a library such as pandas. The `read excel () ` method in pandas can be utilized by providing the file path, sheet name, and any additional pertinent parameters. Take the data Data Frame and extract the labels and features. Determine which column (input variable) represents the features and which column (output variable) represents the labels. Split the Data Frame into X (features) and y (labels) accordingly. Split the feature and label data (X and y) into separate training and test sets using a technique like `train_test_split () ` from the scikit-learn library. This division allows you to have data for training and separate data for evaluating the model's performance. Use Tensor Flow and Keras to build a simple neural network model. Define the architecture of the model by specifying the number of layers and nodes per layer. Based on the needs of your issue, select a suitable activation function and loss function. Assemble the model using assessment metrics and an optimizer. Use the `fit () ` function to fit the neural network model to the training set of data. Indicate the batch size, number of epochs, training data (X_train and y_train), and any other pertinent training parameters. The training data will teach the model linkages and patterns. Take the output out of a certain neural network model layer. The model's learnt features are represented by this output, which is referred to as the seed. Put it into a decision tree model as an input. Using the neural network model's seed as a guide, create and train a decision tree model. Decision tree algorithms, such the `DecisionTreeClassifier`, are available from libraries like scikit-learn. Utilizing the neural network model's retrieved seed, fit the decision tree model.Create a graphical depiction of the decision tree by utilizing packages such as Graphviz and `export_graphviz` from scikit-learn. This plot sheds light on the decision tree's decision-making procedure. For the test data (X_test), use the trained neural network model to produce predictions. Based on the patterns it has learned, the model will either classify or regress the input data. Utilize libraries such as scikit-learn's `confusion_matrix () ` to compute the confusion matrix. The number of accurate and inaccurate predictions is broken down into false positives, false negatives, true positives, and true negatives in the confusion matrix. Determine performance metrics like recall, accuracy, precision, and F1-score by utilizing scikit-learn functions like `classification_report ()}. These metrics evaluate how well the model performs and how well it can classify or regress the data.Make a plot of the Receiver Operating Characteristic (ROC) curve with scikit-learn and matplotlib packages. The true positive rate and false positive rate trade-off in the classification model is illustrated by the ROC curve. Determine the confusion matrix's true negative rate (specificity) and true positive rate (sensitivity or recall). These measures shed light on how well the model distinguishes between good and negative occurrences. The

ratio of accurate predictions to total predictions should be used to calculate the classification accuracy. This score provides an overall assessment of the accuracy of the model.

Examine the confusion matrix data in more detail, paying particular attention to the true positive, false positive, true negative, and false negative rates. These measurements offer in-depth understanding of several facets of model performance. Compute these extra measures to evaluate the model's effectiveness. The percentage of true negative predictions is represented by the true health rate, the percentage of accurately predicted positives by the positive predictive value, and the percentage of true positives among positive predictions by the precision.

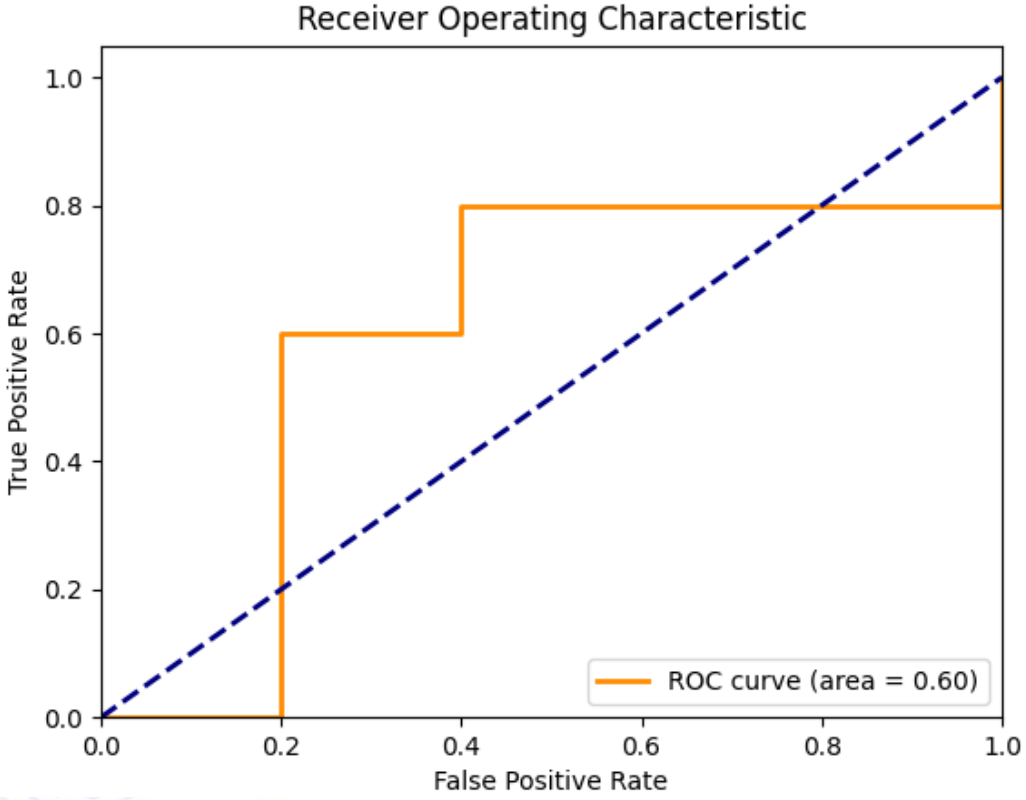

Fig 1. The Receiver Operating Characteristic (ROC)

An illustration that aids in assessing a binary classification model's performance is the Receiver Operating Characteristic (ROC) curve. At different categorization criteria, it shows the true positive rate (TPR) against the false positive rate (FPR). When evaluating and contrasting the performance of various models, including deep learning models, in situations where the cost of misclassifications varies or the class distribution is unbalanced, the ROC curve is frequently employed. A thorough understanding of the model's classification performance at various decision thresholds is offered by the ROC curve. Based on the demands of your particular application, it enables you to assess the trade-off between the genuine positive rate (sensitivity) and the false positive rate (1-specificity). The ROC curve can be used to analyze various points in order to determine how well the model classifies positive cases while reducing false alarms. By comparing and choosing the top-performing model, you may determine which one works best by looking at the ROC curves of each model. In general, models that have curves closer to the upper-left corner of the plot perform better. The classification threshold in binary classification issues establishes the threshold at which expected probabilities are translated into class labels. The ROC curve offers information on the trade-off between TPR and FPR, which aids in the selection of an appropriate threshold. You can modify the threshold to prioritize sensitivity or specificity in accordance with the particular needs of your secure content caching system. The robustness of the model and its capacity to deal with imbalanced datasets or different misclassification costs can be inferred from the ROC curve's form. The model's performance may not be noticeably better than a random guess if the ROC curve is near a diagonal line, which denotes random

categorization. Nevertheless, as seen in fig. 1, a steeper curve with greater TPR and lower FPR denotes a stronger discriminating power.

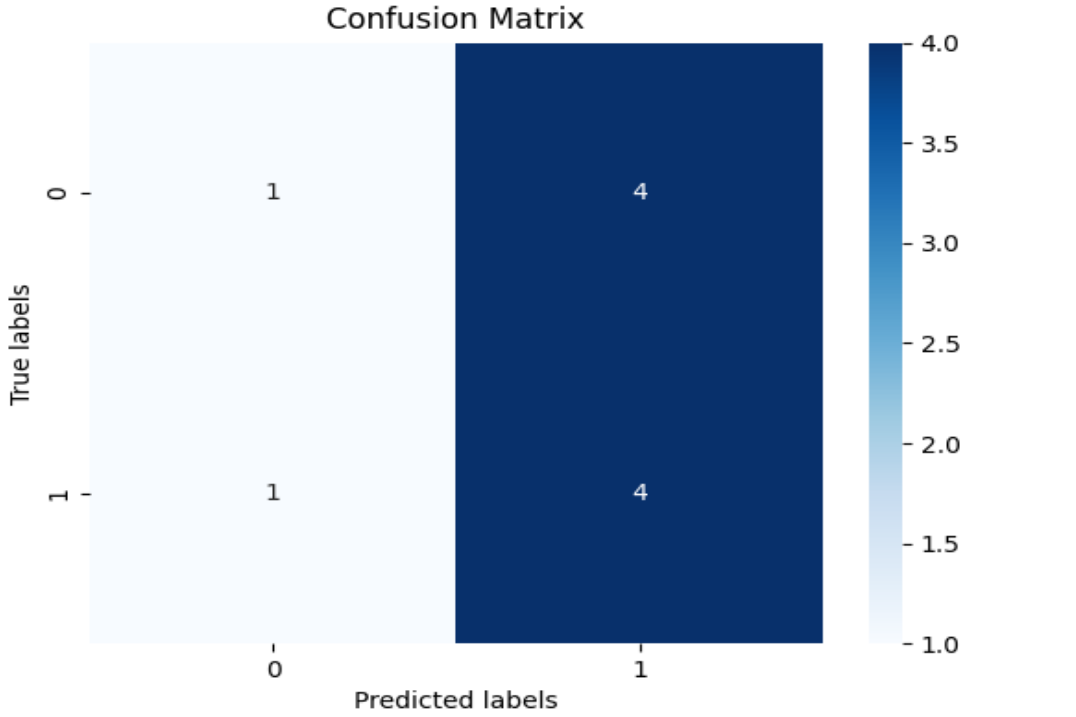

Fig 2. Confusion Matrix

One popular evaluation method for modeling and simulating secure content caching systems in Internet of Things networks is the confusion matrix. It offers a tabular display of the classifications that a classification model actually made and the ones that were predicted. As shown in fig. 2, the Confusion Matrix is composed of four primary parts:

1. True Positives (TP): This is the amount of positive cases that the model accurately identified as positive.
2. True Negatives (TN): This is the amount of negative cases that the model accurately identified as negative.
3. False Positives (FP): Known also as Type I error or false alarms, these are the number of negative events that the model mistakenly identified as positive.
4. False Negatives (FN): Also called Type II errors or misses, these are the number of positive examples that the model misclassified as negative.

The following are some advantages of modeling and simulating secure content caching systems in Internet of Things networks using the Confusion Matrix:

1. Performance evaluation: Beyond just accuracy, the Confusion Matrix offers a more thorough understanding of the model's performance. It enables you to comprehend the various mistakes the model makes, including false positives and false negatives. The distribution of these mistakes can be examined to find possible places where the secure content caching system needs to be improved.
2. Class imbalance assessment: The distribution of the two classes (secure material vs. non-secure information, for example) may be unbalanced in a variety of real-world circumstances, including IoT networks. The Confusion Matrix is a useful tool for evaluating how class imbalance affects the model's functionality. It enables you to determine whether the model struggles to accurately categorize examples from the minority class or is biased towards the majority class.

3. Determining the threshold: A good tool for figuring out the best classification threshold for the model is the confusion matrix. You can manage the ratio of recall (or sensitivity) to precision by varying the threshold. You can select a threshold that reduces false positives or false negatives based on the particular needs of your secure content caching system.
4. Model comparison: By looking at each model's individual performance metrics, including precision, accuracy, F1 score, and recall, the Confusion Matrix makes it simple to compare several models. It is possible to determine which model performs better in terms of reducing the number of false positives, false negatives, or misclassifications overall.

A metric used to assess a classification model's general health is classification accuracy. It is determined by dividing the total number of forecasts by the ratio of accurate predictions (true positives + true negatives). Classification accuracy is useful for evaluating model performance in general, but it does not offer specific information about the kinds of errors the model generates. Conversely, the confusion matrix offers a more thorough evaluation of the model's performance by classifying the actual and projected classifications into four groups: false positives, false negatives, true positives, and true negatives. It makes it possible to comprehend the many kinds of faults the model introduces and how frequently they occur.

Figure 3 illustrates the link between the confusion matrix and classification accuracy:

1. Classification accuracy: these measures how accurate the model's predictions are overall, taking into account all possible kinds of errors. It is determined by dividing the total number of forecasts by the ratio of accurate predictions (true positives + true negatives). Even while classification accuracy is a frequently used metric, it cannot give a clear picture of how well a model is performing, particularly when working with datasets that are imbalanced or when the prices of various error kinds vary.
2. Confusion matrix: Offers a tabular depiction of the model's actual and anticipated classifications. Predictions are separated into four groups: true negatives, true positives, false negatives, and false positives. A more thorough examination of the model's performance is made possible by the confusion matrix, which also enables the assessment of class imbalance, the identification of certain error kinds, and the establishment of ideal thresholds. Confusion matrix calculations aid in the computation of several performance metrics, including F1 score, recall, specificity, and accuracy, which offer more precise insights into the error distribution and model performance. These metrics, which are obtained from the confusion matrix's values, can help identify the model's advantages and disadvantages.

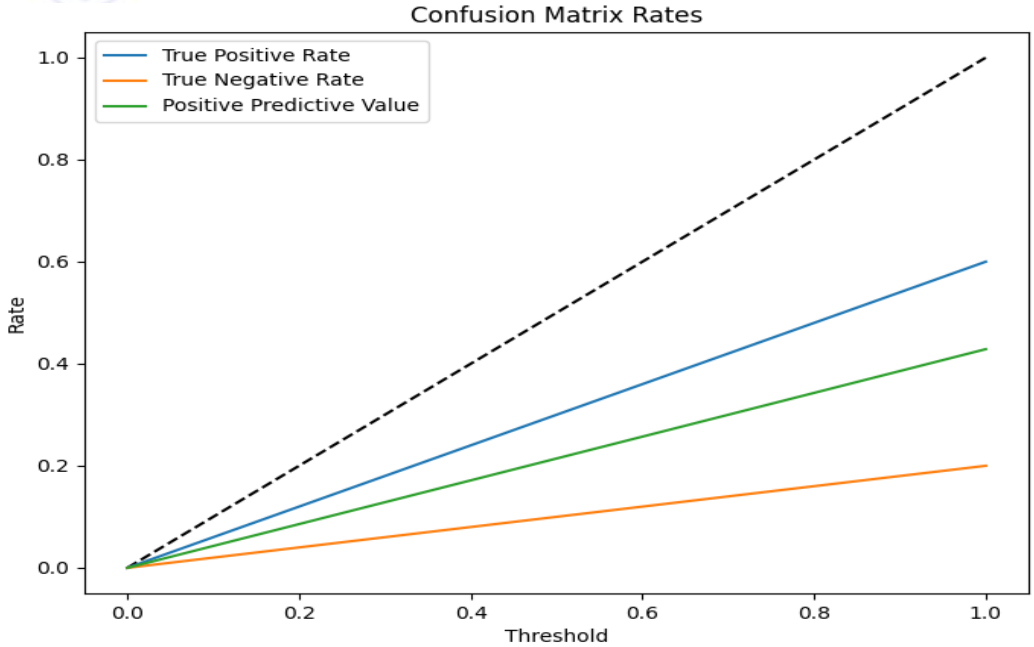

fig 3. Confusion Matrix Rates

Conclusion

Enhancing the security element of the system while optimizing content caching is the goal of designing and simulating a secure content caching system in Internet of Things networks using deep learning techniques like decision trees and neural networks. The optimal content caching technique is chosen using decision trees, taking into account the popularity of the material, network conditions, and security needs. On the other side, through extensive dataset training, neural networks assist in creating decisions that are security-conscious. The procedure entails gathering data, preparing it, choosing a suitable deep learning architecture, training the models, assessing their performance, and making any required improvements. The findings are intended to shed light on the advantages of securing content caching in Internet of Things networks by utilizing deep learning techniques. In this context, evaluation tools like the confusion matrix and receiver operating characteristic (ROC) curve are frequently utilized. . The receiver operating characteristic curve (ROC curve) helps assess a binary classification model's efficacy by illustrating the trade-off between the rate of true positives and the rate of false positives at different classification levels.. You may assess the model's accuracy in classifying positive events while reducing false alarms by looking at the curve. Better performance is indicated by a steeper curve with a greater true positive rate and a lower false positive rate. Based on particular application needs, the ROC curve also aids in choosing an acceptable classification threshold.A tabular representation of the actual and expected classifications made by a classification model is provided by the Confusion Matrix. There are four types of it: true positives, true negatives, false positives, and false negatives. Beyond just accuracy, the Confusion Matrix enables a more thorough performance review. It assists in identifying the many kinds of errors that the model makes, evaluating class imbalance, choosing the best thresholds, and contrasting models according to performance metrics.

A classification model's overall correctness is gauged by its classification accuracy, but by examining the different kinds of errors and how frequently they occur, the Confusion Matrix offers more detailed information about the model's performance.

Reference

1. V. Dastjerdi and R. Buyya, "Fog computing: Helping the Internet of Things realize its potential," Computer, vol. 49, no. 8, pp. 112-116, 2016.
2. Z. Yan, P. Zhang, and A. V. Vasilakos, "A survey on trust management for Internet of Things," Journal of network and computer applications, vol. 42, pp. 120-134, 2014.
3. Shanthamallu, U. S., Spanias, A., Tepedelenlioglu, C., & Stanley, M. (2017, August). A brief survey of machine learning methods and their sensor and IoT applications. In *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)* (pp. 1-8). IEEE.
4. Goyal, S., Sharma, N., Bhushan, B., Shankar, A., & Sagayam, M. (2021). Iot enabled technology in secured healthcare: Applications, challenges and future directions. *Cognitive Internet of Medical Things for Smart Healthcare: Services and Applications*, 25-48
5. Al-Garadi, M. A., Mohamed, A., Al-Ali, A. K., Du, X., Ali, I., & Guizani, M. (2020). A survey of machine and deep learning methods for internet of things (IoT) security. *IEEE Communications Surveys & Tutorials*, *22*(3), 1646-1685
6. Zantalis, F., Koulouras, G., Karabetsos, S., & Kandris, D. (2019). A review of machine learning and IoT in smart transportation. *Future Internet*, *11*(4), 94.
7. M. Abomhara, "Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks," Journal of Cyber Security and Mobility, vol. 4, no. 1, pp. 65-88, 2015.
8. D. Serpanos, "The Cyber-Physical Systems Revolution," Computer, vol. 51, no. 3, pp. 70-73, 2018.
9. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," Computer, vol. 50, no. 7, pp. 80-84, 2017.

10. Y. Xin et al., "Machine Learning and Deep Learning Methods for Cybersecurity," IEEE Access, 2018.
11. X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," IEEE access, vol. 2, pp. 514-525, 2014.
12. R. Sfar, E. Natalizio, Y. Challal, and Z. Chtourou, "A roadmap for security challenges in the Internet of Things," Digital Communications and Networks, 2017.
13. F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, "Internet of Things security: A survey," Journal of Network and Computer Applications, vol. 88, pp. 10-28, 2017.
14. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of Things Security: a top-down survey," Computer Networks, 2018.
15. J. Granjal, E. Monteiro, and J. S. Silva, "Security for the internet of things: a survey of existing protocols and open research issues," IEEE Communications Surveys & Tutorials, vol. 17, no. 3, pp. 1294-1312, 2015.
16. B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in Internet of Things," Journal of Network and Computer Applications, vol. 84, pp. 25-37, 2017.