**HACKFEST DECODING COMPLEX INFORMATION SECURITY CHALLENGES**

**CODE ANOMALY DETECTION USING LOGISTIC REGRESSION**

**Introduction and Motivation:**

Anomaly detection is an important task in many fields, including finance, healthcare, cybersecurity, and manufacturing. It involves identifying instances that deviate from normal behaviour or patterns in each dataset. Anomalies can indicate potentially critical events, such as cyberattacks, fraudulent activities, or equipment failures, and can have significant consequences if left undetected.

Traditional approaches to anomaly detection often rely on rule-based methods that require human experts to manually specify rules and thresholds. However, these approaches have limitations in their ability to detect unknown or unexpected anomalies, and require frequent updates and adjustments as new anomalies emerge. Machine learning-based approaches to anomaly detection can offer several advantages over rule-based methods, including higher accuracy, efficiency, and scalability, and the ability to detect previously unseen anomalies.

Machine learning algorithms can automatically learn to identify patterns and anomalies in data without the need for human input. They can identify complex relationships and hidden patterns that may not be apparent to human experts. Additionally, machine learning models can be trained on large datasets, allowing them to capture a wide range of normal and anomalous behaviours, and to adapt to changing environments.

One area where machine learning-based anomaly detection is particularly important is cybersecurity. As cyber-attacks become increasingly sophisticated, traditional rule-based methods for detecting anomalies may not be effective in identifying new and unknown threats. Machine learning-based approaches, on the other hand, can learn to identify unusual network traffic patterns or system behaviours that may indicate a cyber-attack. For example, machine learning models can be trained to identify the characteristic signatures of malware, detect unusual login patterns, or identify unusual network traffic flows.

Another area where machine learning-based anomaly detection is important is in manufacturing, where it can be used to detect equipment failures or identify deviations in production processes. Machine learning algorithms can be trained to identify patterns in sensor data from manufacturing equipment and alert operators to potential issues before they lead to downtime or production delays. This can result in significant cost savings and increased efficiency in manufacturing operations.

In healthcare, machine learning-based anomaly detection can be used to detect medical errors, such as incorrect medication dosages or surgical mistakes. Machine learning models can analyse electronic health records and alert healthcare providers to potential errors or deviations from expected norms. This can improve patient safety and reduce the risk of adverse medical events.

To develop code for anomaly detection using machine learning, several steps are typically involved. First, a dataset of normal behaviour is collected and labelled. This dataset is used to train a machine learning model to identify normal behaviour and detect deviations from it. The model is then evaluated on a separate test dataset to assess its accuracy and performance. Once the model has been trained and validated, it can be used to detect anomalies in new data.

There are several challenges involved in developing code for anomaly detection using machine learning. One challenge is the availability of labelled data. Labelled data is required to train a machine learning model, but collecting and labelling data can be time-consuming and expensive. Additionally, anomalies may be rare events, making it difficult to collect enough labelled anomalies for training.

Another challenge is the potential for false positives and false negatives. False positives occur when the model identifies normal behaviour as anomalous, while false negatives occur when the model fails to detect actual anomalies. Balancing the trade-off between false positives and false negatives is important in developing an effective anomaly detection system.

Additionally, machine learning models can be vulnerable to adversarial attacks, where an attacker deliberately manipulates data to evade detection by the model. This is particularly relevant in cybersecurity, where attackers may attempt to evade detection by modifying their behaviour or network traffic patterns.

In summary, developing code for anomaly detection using machine learning is an important and challenging task that has applications in many domains. Machine learning-based approaches can offer several advantages over traditional rule-based methods, including higher accuracy and efficiency in detecting anomalies. However

**Problem Statement:**

Source code anomaly detection is an important process in software development as it helps to identify and fix errors or security vulnerabilities before they can cause significant damage. Anomaly detection is the process of identifying patterns in data that deviate from the expected or normal behaviour. In the context of source code, anomalies may include errors in the syntax, semantic inconsistencies, or deviations from standard coding practices.

The goal of this project is to develop a Python-based system that can automatically detect anomalies in source code. The system will use machine learning techniques to learn the patterns of normal code behaviour and identify any deviations from this pattern as anomalies.

The first step in this project is to collect a large dataset of source code samples. This dataset should contain a mix of normal and anomalous code examples. Normal code examples will be used to train the machine learning models, while anomalous code examples will be used to evaluate the effectiveness of the system.

Once the dataset has been collected, the next step is to pre-process the data to extract relevant features that can be used to train the machine learning models. Some of the features that can be extracted include the syntax, code structure, variable names, function names, and comments.

Next, various machine learning algorithms will be used to train the models on the pre-processed data. Some of the popular machine learning algorithms that can be used for this task include decision trees, random forests, support vector machines, and neural networks.

After the models have been trained, they can be used to predict whether a given code sample is anomalous or not. The system will take a new code sample as input and extract the relevant features. The machine learning models will then analyse the features and predict whether the code is anomalous or not.

To evaluate the effectiveness of the system, various metrics such as precision, recall, and F1 score will be calculated. These metrics will help to determine the accuracy of the system in detecting anomalies in source code.

Finally, the system will be tested on a real-world dataset to evaluate its effectiveness in detecting anomalies in real-world scenarios. The results of these tests will be used to fine-tune the system and improve its accuracy.

In conclusion, the development of a Python-based system for source code anomaly detection is an important task in software development. The system will help to identify and fix errors and security vulnerabilities in source code, which can save time and resources in the long run. By using machine learning techniques to learn the patterns of normal code behaviour and identify deviations from this pattern as anomalies, the system can help to improve the overall quality and security of software applications.

**Anomaly Detection:**

Anomaly detection is a data analysis technique used to identify unusual patterns or data points that do not conform to the expected behaviour. It is an important technique used in a wide range of applications, including fraud detection, fault detection, intrusion detection, medical diagnosis, and quality control.

In general, anomaly detection involves three key steps: data pre-processing, feature extraction, and anomaly detection.

Data pre-processing involves cleaning and preparing the data for analysis. This may involve removing missing values, scaling the data, and handling outliers. In some cases, the data may need to be transformed into a different format or representation, such as time-series data.

Feature extraction involves identifying relevant features or attributes of the data that can be used to distinguish between normal and anomalous patterns. This may involve selecting a subset of the available features or deriving new features from the existing ones.

Anomaly detection involves applying a statistical or machine learning algorithm to the data to identify anomalous patterns. There are several types of anomaly detection algorithms, including unsupervised, semi-supervised, and supervised methods.

Unsupervised anomaly detection methods do not require labelled data and aim to identify patterns that are significantly different from the normal behaviour. One common unsupervised method is clustering, which groups similar data points together based on their distance or similarity. Anomalous data points are then identified as those that do not belong to any cluster or belong to a small or sparsely populated cluster.

Another unsupervised method is density-based anomaly detection, which identifies anomalous data points as those that have a low probability density compared to the surrounding data points. This can be achieved using methods such as local outlier factor (LOF) or k-nearest neighbours (k-NN).

Semi-supervised anomaly detection methods use a small amount of labelled data to train a model that can identify anomalous patterns in the unlabelled data. One common semi-supervised method is one-class classification, which learns a model of the normal behaviour and identifies anomalous patterns as those that do not fit this model.

Supervised anomaly detection methods require labelled data to train a model that can identify anomalous patterns. This may involve using a classification algorithm to distinguish between normal and anomalous patterns based on a set of labelled examples.

Anomaly detection algorithms can be evaluated using various metrics, such as precision, recall, and F1 score. Precision measures the proportion of true positives among the predicted anomalies, while recall measures the proportion of true positives among all actual anomalies. The F1 score is a harmonic mean of precision and recall and provides an overall measure of algorithm performance.

There are several challenges associated with anomaly detection, including the definition of what constitutes normal behaviour, the selection of relevant features, and the choice of an appropriate algorithm. In addition, the prevalence of anomalies in the data may be low, which can make it difficult to train and evaluate an anomaly detection algorithm.

Despite these challenges, anomaly detection is a powerful technique that can be used to identify unusual patterns and behaviours in a wide range of applications. By detecting anomalies early, it is possible to prevent or mitigate potential problems and improve the overall quality and reliability of systems and processes.

**Python:**

Python is a high-level, general-purpose programming language that is widely used for a variety of tasks, such as web development, data analysis, machine learning, and more. Python is known for its simplicity, readability, and ease of use, making it a popular choice for beginners as well as experienced programmers. In this article, we will explore the various aspects of the Python programming language, including its history, syntax, data types, control structures, functions, modules, and more.

**History of Python:**

Python was created by Guido van Rossum in the late 1980s while working at the National Research Institute for Mathematics and Computer Science (CWI) in the Netherlands. Guido was inspired by the ABC programming language, which was designed to be easy to use and learn. He wanted to create a language that was both easy to use and powerful, and thus, Python was born.

The first version of Python, Python 0.9.0, was released in 1991. Since then, Python has undergone several major revisions and updates, with the latest version being Python 3.11 as of September 2021. Python 2.x was the most popular version of Python for many years, but it has since been deprecated and is no longer being actively developed. Python 3.x is the current version of Python and is the recommended version for new projects.

**Logistic Regression:**

Logistic regression is a statistical method used to model the relationship between a binary dependent variable and one or more independent variables. It is commonly used in machine learning and data analysis to predict the probability of a binary outcome, such as whether a customer will buy a product or not, whether a patient will respond to a treatment or not, or whether a credit card transaction is fraudulent or not.

In logistic regression, the dependent variable is a binary variable that can take on two values, usually 0 and 1. The independent variables, also known as predictors or features, can be either continuous or categorical variables. The goal of logistic regression is to estimate the probability that the dependent variable equals 1 given a set of values for the independent variables.

The logistic regression model is based on the logistic function, also known as the sigmoid function, which maps any real-valued number to a value between 0 and 1. The logistic function has an S-shaped curve, with a steep slope around the value of 0.5, which represents the point at which the probability of the dependent variable equals 1 or 0. The logistic function is defined as:

**= 1 / (1 + exp(-z))**

where p is the probability of the dependent variable equaling 1, z is a linear combination of the independent variables, and exp is the exponential function.

The logistic regression model estimates the parameters of the linear function z using a technique called maximum likelihood estimation. The maximum likelihood estimator finds the values of the parameters that maximize the likelihood of observing the data given the model.

The logistic regression model can be represented as:

**log(p / (1 - p)) = β0 + β1x1 + β2x2 + ... + βkxk**

where p is the probability of the dependent variable equaling 1, x1, x2, ..., xk are the independent variables, β0, β1, β2, ..., βk are the parameters of the model, and log is the natural logarithm.

The logistic regression model estimates the values of the parameters β0, β1, β2, ..., βk using the maximum likelihood estimator. The maximum likelihood estimator finds the values of the parameters that maximize the likelihood of observing the data given the model.

To fit a logistic regression model, the data are typically divided into a training set and a validation set. The training set is used to estimate the parameters of the model, while the validation set is used to evaluate the performance of the model on new data. The performance of the model can be evaluated using various metrics, such as accuracy, precision, recall, and F1 score.

The logistic regression model can be extended to handle more complex relationships between the dependent variable and the independent variables. One common extension is the addition of interaction terms, which represent the product of two or more independent variables. Another extension is the use of polynomial terms, which allow for non-linear relationships between the dependent variable and the independent variables.

Logistic regression can also be used in multi-class classification problems, where the dependent variable can take on more than two values. In this case, several logistic regression models are trained, one for each class, and the predicted class is the one with the highest probability.

Logistic regression has several advantages over other classification algorithms, such as decision trees and support vector machines. It is a simple and interpretable model that can be easily understood and applied in practice. It is also computationally efficient and can handle many independent variables. Additionally, logistic regression can provide measures of uncertainty and significance for the estimated parameters.

However, logistic regression also has some limitations. It assumes that the relationship between the dependent variable and the independent variables is linear and additive. It also assumes that the errors in the model are independent and identically distributed.

**Logistic Regression for Anomaly Detection:**

Logistic regression is a binary classification algorithm that is widely used in machine learning for predicting binary outcomes. It is a supervised learning algorithm that models the probability of the dependent variable (outcome) being 1 or 0, given a set of input features. Anomaly classification, also known as outlier detection, is the task of identifying data points that deviate from the normal behaviour of a system. In this article, we will discuss the use of logistic regression for anomaly classification and whether it is suitable for this task.

**Overview of Anomaly Classification:**

Anomaly classification is an important problem in many domains, including fraud detection, intrusion detection, and fault diagnosis. The goal of anomaly classification is to identify data points that are significantly different from most of the data. Anomalies can be caused by various factors, such as errors in the data, rare events, or malicious behaviour. Anomaly detection is an unsupervised learning task, as it does not require labelled data to train the model.

**Using Logistic Regression for Anomaly Classification:**

Logistic regression is a popular binary classification algorithm that can be used for anomaly classification. The basic idea is to train a logistic regression model on a dataset of normal data and then use the model to predict whether a new data point is normal or anomalous.

The logistic regression model can be trained using the maximum likelihood estimation (MLE) method. MLE is a statistical method that finds the parameters of a model that maximize the likelihood of the observed data. In logistic regression, the MLE method is used to estimate the values of the coefficients (β) that define the relationship between the input features (x) and the output (y).

Once the logistic regression model is trained on normal data, it can be used to predict the probability of a new data point being normal or anomalous. The predicted probability can be compared to a threshold value to determine whether the data point is anomalous or not. If the predicted probability is lower than the threshold value, the data point is classified as anomalous.

Logistic regression has several advantages for anomaly classification:

Simple to implement: Logistic regression is a simple algorithm that can be easily implemented using standard machine learning libraries.

Interpretable: The coefficients of the logistic regression model represent the strength and direction of the relationship between the input features and the output. This makes the model interpretable and helps in understanding the factors that contribute to anomalous behaviour.

Efficient: Logistic regression can be trained quickly on large datasets using optimization algorithms such as gradient descent.

Robust to outliers: Logistic regression is robust to outliers in the training data, as it uses a probabilistic framework to model the relationship between the input features and the output.

Limitations of Logistic Regression for Anomaly Classification

Logistic regression has some limitations for anomaly classification:

Imbalanced data: Anomaly classification datasets are often imbalanced, with a small number of anomalous data points compared to the normal data. Logistic regression may not perform well on imbalanced datasets, as it tends to predict the majority class.

Non-linear relationships: Logistic regression assumes a linear relationship between the input features and the output. If the relationship is non-linear, logistic regression may not be able to capture it.

Overfitting: Logistic regression can overfit the training data if the model is too complex or if the dataset is small. Overfitting can lead to poor generalization performance on new data.

**Pandas Library:**

Pandas is a powerful and popular open-source data manipulation library for Python. It provides data structures and functions to easily work with structured data, such as spreadsheets and relational databases. Pandas can be used for data analysis, data cleaning, and data visualization. In this article, we will discuss the key features of pandas in detail and how to use it for data manipulation.

Data Structures in Pandas

Pandas provides two main data structures for working with structured data:

Series: A one-dimensional labelled array that can hold any data type. A series is similar to a column in a spreadsheet.

Data Frame: A two-dimensional labelled data structure with columns of potentially different data types. A data frame is similar to a spreadsheet or SQL table.

Series

A Pandas Series is a one-dimensional labelled array that can hold any data type. It is like a column in a spreadsheet or a one-dimensional NumPy array. The Series object consists of two arrays, one holding the data values and the other holding the labels, or index.

To create a Series object, we can pass a list of values to the Series constructor. For example, to create a Series of integers, we can do the following:

**Sklearn Library:**

The scikit-learn (sklearn) library is an open-source machine learning library for the Python programming language. It is designed to provide a simple and efficient tool for data analysis and machine learning tasks. Sklearn provides a wide range of algorithms for classification, regression, clustering, and dimensionality reduction. It also includes tools for model selection, data preprocessing, and data visualization.

Here is a brief overview of the main features of the sklearn library:

Classification: Sklearn provides several algorithms for classification, such as logistic regression, k-nearest neighbors, decision trees, random forests, and support vector machines (SVM). These algorithms can be used to predict the class of a new data point based on a set of input features.

Regression: Sklearn also provides several algorithms for regression, such as linear regression, polynomial regression, and support vector regression (SVR). These algorithms can be used to predict a continuous value based on a set of input features.

Clustering: Sklearn provides several algorithms for clustering, such as k-means, hierarchical clustering, and DBSCAN. These algorithms can be used to group similar data points together based on their features.

Dimensionality reduction: Sklearn provides several algorithms for dimensionality reduction, such as principal component analysis (PCA), linear discriminant analysis (LDA), and t-distributed stochastic neighbor embedding (t-SNE). These algorithms can be used to reduce the number of input features while retaining the most important information.

Model selection: Sklearn provides tools for model selection, such as cross-validation and grid search. These tools can be used to select the best model and its hyperparameters for a given task.

Data pre-processing: Sklearn provides tools for data preprocessing, such as scaling, normalization, and feature extraction. These tools can be used to prepare the data for analysis and to improve the performance of the models.

Data visualization: Sklearn provides tools for data visualization, such as scatter plots, histograms, and heatmaps. These tools can be used to explore the data and to visualize the results of the analysis.

Sklearn is easy to use and well-documented, making it a popular choice for both beginners and experts in machine learning. It has a clear and consistent API, which makes it easy to switch between different algorithms and to experiment with different techniques. It also has a large and active community, which provides support, tutorials, and examples.

**Matplotlib:**

Matplotlib is a popular data visualization library for Python. It is used to create high-quality graphs, charts, and other visualizations to help you communicate your data analysis results in a clear and effective manner. This library was first released in 2003, and it has since become one of the most widely used visualization tools in the scientific community. In this article, we will cover all the important aspects of Matplotlib, including its history, installation, and usage.

**History of Matplotlib**

Matplotlib was created by John D. Hunter in 2003, with the goal of creating a plotting library that was like the plotting capabilities of MATLAB, a popular numerical computing software. Hunter named the library Matplotlib, which is short for "MATLAB-like plotting library". He wrote the original code for Matplotlib in Python, and he released it as an open-source project.

Matplotlib quickly gained popularity in the scientific community because of its flexibility and ease of use. Since its initial release, Matplotlib has undergone several major revisions and updates. Today, it is maintained by a team of developers who continue to add new features and improvements.

**Source Code:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Load the dataset

data = pd.read\_csv("data.csv")

data.head()

# Dropping the unwanted columns

data = data.drop(['uniq\_Op numeric', 'uniqOpnd numeric','total\_Op numeric',' total\_Opnd numeric',  ' branchCount numeric'], axis=1)

data.info()

# X and y values

X = data.drop(' defects',axis=1)

y = data[" defects"]

X, y

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.drop(' defects', axis=1), data[' defects'],  test\_size=0.2)

# Normalize the features using standardization

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train a logistic regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

predictions

Predictions output: array([False, False, False, ..., False, False, False])

# Make predictions on the testing set

y\_pred = model.predict(X\_test)

# Calculate the accuracy score and percentage

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

accuracy\_percentage = accuracy \* 100

print('Accuracy:', accuracy\_percentage, '%')

# Plot the accuracy score

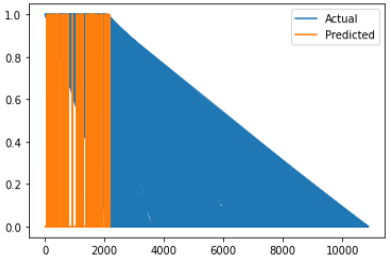
plt.plot(y\_test, label='Actual')

plt.plot(y\_pred, label='Predicted')

plt.legend()

plt.show()

output: Accuracy: 81.85576481396417 % -- Logistic Regression using Standard Scalar



from sklearn.preprocessing import MinMaxScaler

# Normalize the data

scaler = MinMaxScaler()

X\_train\_normalized = scaler.fit\_transform(X\_train)

X\_test\_normalized = scaler.transform(X\_test)

# Train the logistic regression model

model = LogisticRegression()

model.fit(X\_train\_normalized, y\_train)

# Evaluate the model accuracy on the test set

accuracy = model.score(X\_test\_normalized, y\_test)

print(f"Accuracy: {accuracy}")

output: Accuracy: 0.8171796049609554 -- Logistic Regression using MinMaxScalar

# Prediction System

# Get the input from the user to make predictions

input\_data = []

for feature in data.columns[:-1]:

 value = input(f"Enter the value of {feature}: ")

input\_data.append(float(value))

input\_data = scaler.transform([input\_data])

# Make predictions on the input data

prediction = model.predict(input\_data)[0]

if prediction == 0:

    print("There is no anomaly")

else:

    print("There is anomaly.")

# Model to use in the web

import pickle

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = model.predict(X\_test)

# Calculate the accuracy score and percentage

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

accuracy\_percentage = accuracy \* 100

print( accuracy\_percentage)

pickle.dump(model, open('anomaly\_model.sav','wb'))

# Testing the saved model

import pickle

# Load the model from the .sav file

with open('/content/anomaly\_model.sav', 'rb') as f:

  model = pickle.load(f)

def prediction\_model(loc,v,ev,iv,n,v1,l,d,i,e,b,t,loCode\_numeric,loComment\_numeric,loBlank\_numeric,locCodeAndComment\_numeric):

input\_data = [loc,v,ev,iv,n,v1,l,d,i,e,b,t,loCode\_numeric,loComment\_numeric,loBlank\_numeric, locodeAndComment\_numeric]

  input\_data = scaler.transform([input\_data])

  prediction = model.predict(input\_data)[0]

  if prediction == 0:

      print("There is no anomaly")

  else:

      print("There is anomaly.")

# Testing with the custom input values

prediction\_model(1.1,1.4,1.4,1.4,1.3,1.3,1.3,1.3,1.3,1.3,1.3,1.3,2,2,2,2)

# Output

**There is no anomaly**

**Scalability:**

The scalability of anomaly detection code using Python's logistic regression will depend on several factors, including the size of the dataset, the complexity of the model, and the available computational resources.

Logistic regression is a linear model that works well with large datasets and is relatively computationally efficient. However, the scalability of the algorithm may be limited by the number of features in the dataset. As the number of features increases, the logistic regression model may become more complex, requiring more computational resources to train and evaluate.

In addition, the scalability of the anomaly detection code will depend on the implementation details of the code. For example, using vectorized operations and parallel processing techniques can help to speed up the execution of the code and improve scalability.

Overall, logistic regression can be a scalable approach to anomaly detection in Python, but the scalability will depend on the specific details of the dataset and implementation. It may be necessary to explore alternative algorithms or optimization techniques if scalability becomes a bottleneck.

**Conclusion:**

In conclusion, anomaly detection is a crucial task in many fields such as fraud detection, network intrusion detection, and predictive maintenance. With the increasing amount of data available, anomaly detection has become an increasingly important task in data analysis. Python provides numerous libraries and tools for data analysis, including machine learning libraries for anomaly detection. Logistic regression is a simple yet powerful technique for anomaly detection, which can be easily implemented using Python.

Overall, logistic regression is a powerful and flexible algorithm that can be used for anomaly detection in various applications. Python provides a convenient and efficient platform for implementing logistic regression models and evaluating their performance. With the increasing availability of data and the growing importance of anomaly detection in various fields, logistic regression is likely to remain a valuable tool for data analysts and machine learning practitioners in the future.