Dydaktyka Informatyki 20(2025)
ISSN 2083-3156; e-ISSN 2543-9847
DOI: 10.15584/di.2025.20.17
https://www.journals.ur.edu.pl/di

Uniwersytet Rzeszowski

Jacek WOŁOSZYN[1], **Michał WOŁOSZYN**[2]

[1] *ORCID: 0000-0003-4340-9853. Dr inż., Uniwersytet Radomski, Wydział Informatyki i Matematyki, Katedra Informatyki, ul. Malczewskiego 20A, 26-600 Radom, e-mail: jacek.woloszyn@uthrad.pl*
[2] *BSc, student,Goldsmiths, University of London, 8 Lewisham Way, London SE 14 6NW, e-mail:mwolo001@gold.ac.uk*

data złożenia tekstu do Redakcji DI: 18.05.2025; data wstępnej oceny artykułu: 27.05.2025

PRACTICAL IMPLEMENTATION OF ARTIFICIAL INTELLIGENCE IN CYBERSECURITY, ONE-CLASS SVM FOR ANOMALY DETECTION IN NETWORK TRAFFIC

PRAKTYCZNA IMPLEMENTACJA SZTUCZNEJ INTELIGENCJI W CYBERBEZPIECZEŃSTWIE, ONE-CLASS SVM DO WYKRYWANIA ANOMALII W RUCHU SIECIOWYM

Keywords: Python, artificial intelligence, cybersecurity, One-Class SVM.
Słowa kluczowe: Python, sztuczna inteligencja, cyberbezpieczeństwo, One-Class SVM.

Abstract

The presented material provides a detailed discussion on the implementation of One-Class SVM in Python, including code examples and a sample CSV file containing network flow parameters such as duration, number of packets, and packet sizes. This is a continuation of the article *Theoretical Considerations on Artificial Intelligence and Cybersecurity: One-Class SVM for Anomaly Detection in Network Traffic*. The authors emphasize the necessity of removing the label column during training, as One-Class SVM is designed to identify anomalous observations based solely on a dataset of normal behavior. The text outlines the key stages of working with the model, including data loading, splitting into training and test sets, scaling, model initialization, and evaluation of results using metrics such as Precision, Recall, and F1-score. It is noted that model evaluation in laboratory conditions may be misleading if only a small number of samples are available. The article also discusses hyperparameter tuning (nu, gamma) and explores potential extensions, including combining One-Class SVM with other algorithms, integration with SIEM systems, and the implementation of real-time streaming data processing.

Streszczenie

W artykule szczegółowo omówiono implementację One-Class SVM w języku Python wraz z przykładami kodu i przykładowym fragmentem pliku CSV, w którym zapisano parametry dotyczące przepływów sieciowych (takie jak czas trwania, liczba pakietów, rozmiary pakietów). Jest to kontynuuacja artykułu *Theoretical Considerations on Artificial Intelligence and Cybersecurity, One-Class SVM for Anomaly Detection in Network Traffic.* Autorzy zwracają uwagę na konieczność usunięcia kolumny etykiet (label) podczas treningu, ponieważ One-Class SVM przystosowany jest do identyfikowania nietypowych obserwacji, bazując wyłącznie na zbiorze zachowań normalnych. W tekście opisano podstawowe etapy pracy z modelem: wczytanie danych, podział na zbiór treningowy i testowy, skalowanie, inicjalizację modelu oraz ewaluację wyników z wykorzystaniem miar typu Precision, Recall czy F1-score. Zwrócono uwagę, że ocena jakości modelu w warunkach laboratoryjnych może być myląca, jeśli dysponuje się jedynie niewielką liczbą próbek. Omówiono także zagadnienie dostrajania hiperparametrów (nu, gamma) i opisano możliwe rozszerzenia obejmujące łączenie One-Class SVM z innymi algorytmami, integrację z systemami SIEM czy wprowadzenie przetwarzania strumieniowego w czasie rzeczywistym.

Introduction

In recent years, there has been a growing interest in artificial intelligence algorithms and machine learning methods that enhance anomaly detection in network traffic, thereby strengthening the security of telecommunication and information systems.

Unsupervised learning-based solutions, such as One-Class SVM, have gained particular importance. These models can be trained solely on “normal” examples, eliminating the need for a large labeled dataset of attacks.

Given that real-world data is often highly diverse (with numerous features, different protocols, and large traffic volumes), proper analytical environment preparation is essential, including data preprocessing and feature scaling.

Additionally, parameter selection, such as nu and gamma, plays a crucial role in balancing effective anomaly detection while avoiding an excessive number of false alarms.

1. Implementation in Python

The implementation in Python[1] typically begins with importing the necessary libraries: NumPy and Pandas for data handling, Scikit-learn (sklearn) for machine learning, and optionally Matplotlib for basic visualization of results.

[1] Ngo Quang-Hung, et al., *Unsupervised Deep Learning for Real-Time Cyber Anomaly Detection*, IEEE Access, Vol. 7, 2019.

Once the environment is set up, the next step is loading the dataset, for example, from a CSV file, where each column contains features describing network traffic. In Python, this can be done using data = pd.read_csv("network_data.csv"), assuming that the file is in CSV format. If the dataset includes a label column (e.g., "label" indicating whether a sample is an attack or normal behavior), it should be ignored during training since One-Class SVM[2] is an unsupervised model and only requires examples of normal traffic. This can be achieved by creating a feature matrix X = data.drop ("label", axis = 1, errors = "ignore"), ensuring that training proceeds without supervision.

To simplify training and evaluation, the data is typically split into training and test sets using train_test_split from Scikit-learn. The following command allocates 80% of the data to training and 20% to testing X_train, X_test = train_test_split (X, test_size = 0.2, random_state = 42). Since many machine learning algorithms, including One-Class SVM, perform better on standardized or normalized data, feature scaling is recommended. Using StandardScaler(), we first fit the scaler on the training data and then apply it to both training and test sets scaler = StandardScaler(); X_train_scaled = scaler.fit_transform (X_train); X_test_scaled = scaler.transform (X_test). This ensures that both datasets have the same range, preventing model bias towards features with large-scale differences.

Once the data is ready, we proceed with training the One-Class SVM model. The Scikit-learn library provides a OneClassSVM class, where key parameters such as kernel = "rbf", nu = 0.01, and gamma = "scale" can be defined. The model is initialized and trained using oc_svm = OneClassSVM (kernel = 'rbf', nu = 0.01, gamma = 'scale'); oc_svm.fit (X_train_scaled). The nu parameter controls the proportion of samples classified as anomalies, while gamma = "scale" allows automatic kernel parameter selection based on the number of features and variance of the data.

To predict anomalies on the test dataset, we execute y_pred = oc_svm.predict (X_test_scaled), where results are returned as +1 (normal sample) or –1 (anomaly). In practice, this output is often converted into a clearer format, for example, anomaly = (y_pred = –1).astype(int), so that in the anomaly vector, 1 represents detected anomalies. If the dataset contains some labeled attack samples, model performance can be evaluated using Precision, Recall, and F1-score. Assuming that the true labels are stored in the "label" column within X_test, we retrieve them using y_test = data.loc [X_test.index, "label"] and

[2] I. Goodfellow, P. McDaniel, N. Papernot, *Making Machine Learning Robust Against Adversarial Inputs*, "Communications of the ACM" 2018, Vol. 61, No. 7; I. Sarker, et al., *IntrudTree: A Machine Learning Based Cyber Security Intrusion Detection Model*, "Symmetry" 2021, Vol. 13, No. 4.

compute the classification report using print(classification_report (y_test, anomaly)).

In typical One-Class SVM applications, attack labels are not always available, so expert knowledge, network segmentation analysis, and security logs are used to verify whether the detected anomalies are truly suspicious. After obtaining initial results, hyperparameter tuning is performed.[3] Adjusting nu influences the number of samples classified as anomalies–higher nu makes the model stricter, while tuning gamma controls whether the decision boundary is tightly fitted to local patterns or kept smooth. These adjustments can be conducted iteratively or via GridSearchCV or RandomizedSearchCV, although One-Class SVM can be computationally expensive when dealing with large datasets.

For deeper analysis, visualizations can be used, particularly with scatter plots highlighting detected anomalies in low-dimensional feature spaces (2-3 features). While network traffic datasets typically have many attributes, making visualization challenging, such simple plots can initially indicate whether the model behaves as expected.

The final step is deployment, transferring the trained model to a production environment, where it can process new log data in batches every few minutes or in near real-time using Kafka and Spark Streaming. Python-based implementation allows not only model training and analysis but also integration with other security mechanisms.[4] If anomalies are detected, alerts can be automatically generated in SIEM systems, or suspicious IP traffic can be blocked, assuming the anomaly score is high enough to minimize false alarms.

2. Example implementation

Below is a sample excerpt from the network_data.csv file containing network traffic data. This is a fictional example that illustrates the possible structure and typical columns that may be present in such a dataset. In practice, the file may contain more rows and additional columns, depending on the analysis requirements.

[3] B. Al-Duwairi, A. Razaque, *A Survey of AI-Based Intrusion Detection Systems for Cloud Computing Environments*, "International Journal of Cloud Applications and Computing" (IJCAC) 2019, Vol. 9, No. 3; Y. Mirsky, et al., *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*, Network and Distributed System Security (NDSS) Symposium, 2018; S. Shalev-Shwartz, S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2019.

[4] N. Kshetri, *Artificial Intelligence in Cyber Security*, "IT Professional" 2018, Vol. 20, No. 3; R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (2nd Edition), MIT Press, 2018.

```
    timestamp,source_ip,dest_ip,source_port,dest_port,protocol,flo
w_duration,packet_count,avg_packet_size,label
    2025-03-01
10:15:22,192.168.1.10,172.217.18.14,49512,80,TCP,123,12,84,0
    2025-03-01
10:15:23,192.168.1.10,172.217.18.14,49512,80,TCP,126,14,82,0
    2025-03-01
10:16:05,192.168.1.15,8.8.8.8,35022,53,UDP,55,8,120,1
    2025-03-01
10:16:10,192.168.1.15,8.8.4.4,35022,53,UDP,60,7,110,1
    2025-03-01
10:18:47,192.168.1.20,10.0.0.5,50001,443,TCP,98,10,90,0
```

Column Overview

timestamp – The timestamp indicating when the given flow or packet was recorded.

source_ip – The IP address of the sender (host or device from which the traffic originates).

dest_ip – The IP address of the recipient (host or target server).

source_port – The source port used for communication.

dest_port – The destination port to which the traffic is directed.

protocol – The transport protocol, such as TCP or UDP (other protocols may appear in certain datasets).

flow_duration – The total duration of the given flow (measured in seconds, milliseconds, or another unit, depending on the tool used).

packet_count – The total number of packets in the given flow.

avg_packet_size – The average packet size (in bytes) in the flow; sometimes, a total byte count column (e.g., "total_bytes") is present instead.

label – The label used for classification; typically,

0 indicates normal traffic

1 indicates a potential anomaly or attack

In real-world environments, network traffic data may contain significantly richer information, such as TCP flags, session identifiers, event types, or metadata specific to a given system (e.g., device type, application name).

If One-Class SVM is used in a fully unsupervised manner, the label column is not used during training. However, it is valuable for evaluation purposes, helping verify whether the anomalies detected by the model actually correspond to attack instances.

```
    import numpy as np
    import pandas as pd
```

```
    from sklearn.svm import OneClassSVM
    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    # Przykładowy wczytanie danych
    data = pd.read_csv("network_data.csv")  # przykładowy plik CSV
    # Załóżmy, żemamykolumny: [feature1, feature2, ..., featureN]
    X = data.drop(["label"], axis=1, errors="ignore")  # jeśli
jest kolumna 'label', ignorujemy ją w unsupervised
    # Podział na zbiór treningowy i testowy
    X_train, X_test = train_test_split(X, test_size=0.2,
random_state=42)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Inicjalizacja modelu z wybranymi parametrami
    oc_svm = OneClassSVM(kernel='rbf', nu=0.01, gamma='scale')
    # Trening na danych przedstawiających "normalny" ruch
    oc_svm.fit(X_train_scaled)
    # Predykcja na danych testowych
    # Zwraca 1 dla normalnych punktów, -1 dla anomalii
    y_pred = oc_svm.predict(X_test_scaled)
    # Konwersja do binarnej postaci, np. anomalia: 1, normalne: 0
    anomalia = (y_pred == -1).astype(int)
    print("Liczba wykrytych anomalii:", sum(anomalia))
    if "label" in data.columns:
    y_test = data.loc[X_test.index, "label"]  # 0 lub 1
        # Tu możnawyliczyć np. precision, recall, F1-score
        from sklearn.metrics import classification_report
        print(classification_report(y_test, anomalia))
```

Sample results

```
    Liczba wykrytych anomalii: 1
    precision    recall  f1-score   support
               0       1.00      0.50      0.67         2
               1       0.00      0.00      0.00         0

        accuracy                           0.50         2
       macro avg       0.50      0.25      0.33         2
    weighted avg       1.00      0.50      0.67         2
```

Discussion of Results

In this example, the classification report shows that there are two samples in the test set (support = 2), meaning that after the random split, we have two observations for evaluation. Since test_size = 0.2, and the dataset contains only five rows, the test set may contain either one or two observations, depending on the rounding and random selection mechanisms. This is typical for very small datasets, where it is important to understand that statistical metrics (such as precision, recall, etc.) are only for illustrative purposes.

Summary of Results

Number of detected anomalies: 1

The One-Class SVM model identified one observation in the test set as an anomaly. In the y_pred vector, this corresponds to a value of –1, which after conversion to the anomaly vector becomes 1, indicating an anomaly for that particular row.

In a real-world scenario, this means that one sample significantly deviated from the learned pattern of normal network traffic, according to the model.

Classification Report (Precision, Recall, F1-score)

Precision indicates what percentage of the samples classified as anomalies by the model are actually anomalies according to the labels.

Recall measures how many of the actual anomalies in the dataset were successfully detected by the model.

F1-score is the harmonic mean of precision and recall, often used for overall model evaluation.

Support represents the total number of occurrences of each class in the test set (in this case, 2 samples labeled as 0 and 0 samples labeled as 1).

Interpretation of the Report

For Class 0 (normal behavior)

Precision = 1.00 → All samples classified as normal by the model were indeed normal according to the labels.

Recall = 0.50 → Only half of the actual normal samples in the test set were correctly identified as normal. This means that one normal sample was incorrectly classified as an anomaly.

F1-score = 0.67 → A combined measure of precision and recall.

For Class 1 (attack, anomaly):

Precision = 0.00, Recall = 0.00 → Since there were no actual anomalies in the test set (support = 0), the model had no opportunity to correctly classify Class 1.

Accuracy = 0.50 → The model correctly classified 1 out of 2 cases, resulting in 50% accuracy. This is a low accuracy, but with such a small number of observations, it is difficult to obtain meaningful statistical results.

Challenges with a Very Small Test Sample

With only 5 rows in total and 2 in the test set, metrics such as accuracy, precision, recall, and F1-score are highly unstable and should not be used to draw strong conclusions.

In a real-world scenario, hundreds or thousands of samples are typically available, allowing for a more reliable performance assessment.

Interpretation for Cybersecurity

One sample was classified as an anomaly, even though according to the label, it was not an attack. This represents a false positive (false alarm).

If the test set contained an actual attack sample (label = 1), the model could either detect it correctly (true positive) or miss it (false negative). However, in this small dataset, no labeled attacks are present for verification.

Need for Further Model Tuning

The default parameters (nu = 0.01, gamma = 'scale') may not be optimal for this dataset.

With a larger dataset, a systematic Grid Search or Randomized Search could be conducted to experiment with different values such as nu = 0.05, nu = 0.1, etc., to find the best balance between anomaly detection performance and false alert reduction.

Importance of Proper Data Splitting

Due to the random split settings (random_state = 42, test_size = 0.2), the test set may not contain all behavior types (both normal and anomalous samples) in proportions representative of the real-world environment.

For such a small dataset, a cross-validation (CV) approach might be more appropriate, although One-Class SVM inherently complicates standard k-fold validation (especially since it does not use labels).

Final Takeaway

In this illustrative example, One-Class SVM classified one sample as an anomaly, even though the label suggested otherwise. This highlights the importance of:

Tuning hyperparameters to reduce false positives,

Using a sufficiently large dataset to obtain meaningful evaluation metrics,

Validating results with expert knowledge and real-world security logs to confirm whether detected anomalies correspond to actual cyber threats. With such a small data size, this is not surprising. In practice, to verify the true effectiveness of the model, a much larger test set would be needed, consisting of many normal samples and a number of real attacks. Only then can the

performance of One-Class SVM be accurately assessed through analyses of measures such as precision, recall, f1-score or AUROC (Area Under the Receiver Operating Characteristic curve).

3. Optimization and Parameter Selection

Optimizing and fine-tuning hyperparameters in One-Class SVM primarily involves balancing two key aspects: accurate anomaly detection and minimizing false alarms. One of the most critical parameters is ν (nu), which determines the proportion of examples that can be treated as outliers. A higher ν value makes the model stricter, meaning it is more likely to classify observations as anomalies. This may improve sensitivity to rare, unusual events, but at the same time, it increases the risk of excessive false alarms.

Another important parameter is gamma (γ) when using the RBF kernel. This coefficient controls how much individual samples influence local distributions in the feature space. A higher gamma value makes the decision boundaries more sensitive to local fluctuations, which may improve precision in detecting specific types of attacks, but can also cause overfitting to random noise in the training set. In practice, the goal is to find an optimal compromise where the model is sensitive enough to detect malicious behavior but does not produce too many false positives.

To select these parameters, cross-validation (CV) is often used, though in One-Class SVM, this can be challenging due to the lack of clear labels in the training samples. One approach is to inject synthetic anomalies into part of the data, introducing clearly abnormal events and verifying whether the model correctly classifies them as deviations. Alternatively, if a small dataset of known attacks is available, it can be used solely as a test base, while the model remains trained on a purely normal dataset.

Tools such as GridSearchCV or RandomizedSearchCV play an essential role in parameter tuning, although computational costs must be considered, as One-Class SVM can be computationally intensive for large datasets. In such cases, random sampling or dimensionality reduction (e.g., PCA) can be applied before iterative tuning to reduce complexity.

Another crucial aspect is observing long-term trends. In production environments, conditions evolve dynamically–new devices appear, network traffic patterns shift, and new attack types emerge. A One-Class SVM model trained on historical data may require periodic retraining or continuous updates to remain effective.

If the system includes automated validation mechanisms (such as monitoring the number of alarms per time period or tracking the ratio of confirmed incidents in SIEM logs), this information can guide decisions on whether parameters (e.g., ν) need adjustment, and whether the entire training process should be repeated.

Finally, it is important to note that One-Class SVM's performance heavily depends on data quality and structure. Therefore, hyperparameter tuning must often be combined with feature engineering–removing highly correlated attributes, defining new features that enhance the distinction between normal and abnormal traffic, or transforming data to emphasize critical patterns (e.g., separating inbound vs. outbound traffic).

This entire sequence–from defining evaluation strategies, iteratively testing different ν and γ values, to monitoring and potential retraining–ensures that the model remains a reliable tool for anomaly detection in a constantly evolving network environment.

4. Extensions

Extensions cover both combining One-Class SVM with other anomaly detection methods and integrating the solution into larger security management systems (SIEM) or real-time network traffic processing frameworks.

The first approach often involves ensemble learning, which combines multiple machine learning algorithms, such as Isolation Forest, autoencoders, or clustering methods (e.g., DBSCAN). The idea is that certain types of attacks may be better detected by specific models (One-Class SVM may effectively capture unusual deviations, while Isolation Forest might be better at handling varied traffic statistics). By aggregating results from multiple sources, both sensitivity and specificity of anomaly detection improve, reducing false alarm rates in real-world environments.

The second extension involves integration with SIEM (Security Information and Event Management) systems, such as Splunk or IBM QRadar, which collect all network alerts and security events. Implementing One-Class SVM in such an environment allows periodic analysis of network or system log data, and if the model detects a significant deviation, it generates an incident suspicion alert. SIEM systems, which also aggregate information from other sources (e.g., firewalls, IDS systems, application servers), can correlate various events to determine whether a suspicious observation has serious security implications or is simply a harmless anomaly.

Integration with SIEM also enables automated defensive responses, ranging from blocking incoming traffic from suspicious IPs to advanced measures such as dynamically switching network segments or isolating machines that exhibit highly abnormal activity.

The third extension area is stream processing, meaning real-time data analysis[5] (almost immediately after it appears in the infrastructure). Solutions such as Apache Kafka, Spark Streaming, or Apache Flink allow a continuous flow of incoming network traffic records, eliminating the need to wait for periodic anomaly detection cycles (e.g., hourly or daily scans).

With one or more One-Class SVM models, teams can implement an architecture where data is rapidly scaled in a computing cluster, and the anomaly detection results are sent back to SIEM or fed into automated response mechanisms.

Deploying such functionality in production environments requires careful resource planning (CPU and memory efficiency), as live network flow analysis typically generates large volumes of information. However, the benefits of real-time threat detection and mitigation can significantly enhance an organization's overall security.

Combining One-Class SVM with a streaming architecture is thus a natural extension of anomaly detection and enables faster detection and response to security incidents.

Summary

The discussed One-Class SVM method proves to be highly useful in dynamic and unpredictable network environments, where new attack types emerge too rapidly for traditional signature-based systems to detect them effectively.

The presented implementation steps confirm that the key to success lies in both a well-prepared dataset (in terms of quality, format, and scaling) and the proper selection of model hyperparameters. This allows for an effective distinction between normal traffic and potentially dangerous deviations, enabling rapid response to detected incidents.

However, in production deployment, it is crucial to consider evolving threats, ensuring that the model is regularly updated and its performance continuously monitored.

---

[5] Chen, Yi, Ding, Wang. *Isolation Forest for Anomaly Detection*, Mathematical Problems in Engineering, 2018; J.L. Leevy, T.M. Khoshgoftaar, R.A. Bauder, N. Seliya,. *A Survey on Addressing High-class Imbalance in Big Data*, "Journal of Big Data", Vol. 5, 2018.

When combined with other detection techniques and SIEM tools, One-Class SVM becomes a valuable component of a multi-layered cybersecurity strategy, allowing organizations to continuously track deviations from normal behavior and detect attacks that do not match any known patterns.

Bibligraphy

Al-Duwairi B., Razaque A., *A Survey of AI-Based Intrusion Detection Systems for Cloud Computing Environments*, "International Journal of Cloud Applications and Computing" (IJCAC) 2019, Vol. 9, No. 3.

Chen Yi, Ding Wang, *Isolation Forest for Anomaly Detection*, Mathematical Problems in Engineering, 2018.

Goodfellow I., McDaniel P., Papernot N., *Making Machine Learning Robust Against Adversarial Inputs*, "Communications of the ACM" 2018, Vol. 61, No. 7.

Kshetri N., *Artificial Intelligence in Cyber Security*, "IT Professional" 2018, Vol. 20, No. 3.

Leevy J.L., Khoshgoftaar T.M., Bauder R.A., Seliya N., *A Survey on Addressing High-class Imbalance in Big Data*, "Journal of Big Data" 2018, Vol. 5.

Mirsky Y., et al., *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*, Network and Distributed System Security (NDSS) Symposium, 2018.

Ngo Quang-Hung, et al., *Unsupervised Deep Learning for Real-Time Cyber Anomaly Detection*, IEEE Access, Vol. 7, 2019.

Shalev-Shwartz S., Ben-David S., *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2019.

Sutton R.S., Barto A.G., *Reinforcement Learning: An Introduction* (2nd Edition), MIT Press, 2018.