

Class 05: Student Guide - Outlier Identification and Anomaly Detection

AI in Cybersecurity Course
Instructor Steve Smith

Class Overview

Topic: Anomaly Detection for Cybersecurity






Duration: 90 minutes

Lab Component: Hands-on coding with malware dataset

Assignment Due: August 5, 11:59 PM ET (15 points)

Learning Objectives

By the end of this class, you should be able to:

-  Explain what anomaly detection is and why it's crucial for cybersecurity
-  Implement three different anomaly detection algorithms using scikit-learn
-  Apply these algorithms to detect potential malware in a real dataset
-  Compare and evaluate different anomaly detection approaches
-  Use Large Language Models (LLMs) to generate Python code for data analysis

Key Concepts

What is Anomaly Detection?

Definition: Identifying data points that deviate significantly from normal behavior patterns.

Why it is important in Cybersecurity:

- Detects unknown threats (zero-day attacks)
- Identifies unusual user behavior
- Monitors system health
- Discovers new attack patterns

Types of Anomalies

1. **Point Anomalies:** Single unusual events
 - Example: One login from an unusual location
2. **Contextual Anomalies:** Unusual in specific context
 - Example: Access during off-hours
3. **Collective Anomalies:** Group of events is unusual
 - Example: Coordinated botnet activity

Anomaly Detection vs. Supervised Learning

Anomaly Detection	Supervised Learning
No labeled "attack" data	Has labeled examples
Looking for outliers	Learning from examples
Unsupervised approach	Supervised approach
Detects unknown threats	Detects known patterns

Three Anomaly Detection Algorithms

1. K-Means Clustering

How it works: Groups similar data points into clusters, anomalies are far from cluster centers.

Pros:

- Simple to understand and implement
- Fast for moderate-sized datasets

Cons:

- Need to choose number of clusters (k)
- Struggles with complex cluster shapes

Code Example:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
# Scale your data first
```

```

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Apply K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
# Find distances to cluster centers
distances = kmeans.transform(X_scaled)

```

2. Isolation Forest

How it works: Builds random trees to isolate data points. Anomalies are easier to isolate (require fewer splits).

Pros:

- Handles high-dimensional data well
- Less sensitive to parameter tuning
- Good performance on large datasets

Cons:

- Can be computationally expensive
- Black box - harder to interpret

Code Example:

```

from sklearn.ensemble import IsolationForest
# Create and fit model
iso_forest = IsolationForest(
    contamination=0.1, # Expected proportion of anomalies
    random_state=42
)
# Predict anomalies (-1 = anomaly, 1 = normal)
anomalies = iso_forest.fit_predict(X_scaled)
# Get anomaly scores
scores = iso_forest.decision_function(X_scaled)

```

3. One-Class SVM

How it works: Creates a boundary around "normal" data points. Anything outside the boundary is anomalous.

Pros:

- Can handle complex decision boundaries
- Works well with non-linear data

Cons:

- Sensitive to parameter tuning
- Can be slow on large datasets
- Requires careful parameter selection

Code Example:

```
from sklearn.svm import OneClassSVM
# Create and fit model
oc_svm = OneClassSVM(
    nu=0.1, # Expected proportion of anomalies
    kernel='rbf',
    gamma='scale'
)
# Predict anomalies (-1 = anomaly, 1 = normal)
anomalies = oc_svm.fit_predict(X_scaled)
```

Student Solo Exercise: Malware Anomaly Detection

Dataset Information

- **File:** `dataset_malwares.csv`
- **Size:** 19,611 samples
- **Features:** 69 different file characteristics
- **Source:** VirusShare database

Step-by-Step Lab Process

1. Data Preparation

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('<insert_your_path_to_dataset_malwares.csv>')
# Explore the data
print(f"Dataset shape: {df.shape}")
print(f"Features: {df.columns.tolist()}")
print(df.head())
# Prepare features (remove labels if present)
X = df.select_dtypes(include=[np.number])
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2. Apply All Three Algorithms

```
# K-Means

kmeans = KMeans(n_clusters=5, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)
# Isolation Forest
iso_forest = IsolationForest(contamination=0.1, random_state=42)
iso_anomalies = iso_forest.fit_predict(X_scaled)
# One-Class SVM
oc_svm = OneClassSVM(nu=0.1, kernel='rbf')
svm_anomalies = oc_svm.fit_predict(X_scaled)
```

3. Compare Results

```
# Count anomalies detected by each method
print(f"K-Means anomalies: {len(kmeans_labels[kmeans_labels == -1])}")
print(f"Isolation Forest anomalies: {len(iso_anomalies[iso_anomalies == -1])}")
print(f"One-Class SVM anomalies: {len(svm_anomalies[svm_anomalies == -1])}")
# Find overlapping anomalies
iso_anomaly_indices = set(np.where(iso_anomalies == -1)[0])
svm_anomaly_indices = set(np.where(svm_anomalies == -1)[0])
overlap = iso_anomaly_indices.intersection(svm_anomaly_indices)
print(f"Overlapping anomalies: {len(overlap)}")
```



Assignment #03: Basic Setup & Data Import



Objective

Apply the anomaly detection techniques learned in Classes 4 and 5 to your preprocessed malware dataset from Assignment #2. This assignment bridges data preprocessing with unsupervised machine learning, preparing you for supervised classification in later assignments.



Requirements

1 Environment Setup & Data Import (3 points)

- Load your preprocessed dataset from Assignment #2
- Import required libraries (scikit-learn, pandas, numpy, matplotlib, seaborn)
- Display dataset summary statistics and confirm preprocessing was successful
- Verify data types and shape are appropriate for ML algorithms

2 LLM-Assisted Code Generation (3 points)

- **Demonstrate LLM prompting skills** by including at least 3 different prompts you used to generate Python code
- Document the prompts in markdown cells before the generated code
- Show how you refined prompts to get better results
- Examples of good prompts:
 - "Generate Python code to implement Isolation Forest for malware detection with contamination parameter tuning"

- "Create a function to visualize anomaly detection results using matplotlib with proper labels and legends"
- "Write code to compare three anomaly detection algorithms and create a summary table of results"

③ Anomaly Detection Implementation (6 points)

Implement all three algorithms covered in Class 5:

K-Means Clustering (2 points)

- Implement K-means with appropriate number of clusters (test $k=2,3,4,5$)
- Calculate distances from cluster centers to identify outliers
- Use elbow method or silhouette analysis to choose optimal k

Isolation Forest (2 points)

- Implement with contamination parameter testing (0.1, 0.2, 0.3)
- Generate anomaly scores for all samples
- Identify samples flagged as anomalies (-1)

One-Class SVM (2 points)

- Implement with ν parameter testing (0.1, 0.2, 0.3)
- Generate decision function scores
- Compare boundary definition approaches

④ Feature Analysis for Cybersecurity Context (2 points)

- Select and analyze the most relevant features for malware detection
- Create correlation heatmap of top 10 features
- Analyze feature importance in the context of PE file characteristics:
 - Entropy measures
 - Import function counts
 - Section characteristics
 - File size metrics

⑤ Results Comparison & Visualization (1 point)

- Create comparison table showing:
 - Algorithm name
 - Number of anomalies detected
 - Percentage of dataset flagged as anomalous
 - Parameter settings used
- Generate visualizations using PCA for 2D representation of anomalies
- Include confusion matrix comparison (if you have ground truth labels)



Deliverables

- **Jupyter notebook** with all code, outputs, and analysis
- **LLM prompts documentation** in markdown cells
- **Visualization outputs** embedded in notebook
- **Summary analysis** of which algorithm performed best and why

Submission Guidelines:

- **Due:** August 5, 11:59 PM ET
- **Format:** Jupyter notebook (.ipynb file)
- **Include:** Code cells with outputs, markdown explanations
- **Naming:** `YourLastName_Assignment03.ipynb`

Example LLM Prompts:

- "Generate Python code using scikit-learn to load a CSV file and perform basic data exploration"
- "Create code to detect anomalies in a dataset using Isolation Forest with scikit-learn"
- "Write Python code to compare the results of multiple anomaly detection algorithms"



Evaluation Challenges

Why is Anomaly Detection Evaluation Difficult?

- **No ground truth:** We don't know what "normal" really is
- **Imbalanced data:** Very few anomalies compared to normal data
- **Subjective definition:** What counts as "anomalous" can vary
- **Context matters:** Same behavior might be normal or anomalous depending on context

Evaluation Strategies:

1. **Expert review:** Have domain experts validate detected anomalies
2. **Known anomalies:** Test on datasets with known anomalies
3. **Stability:** Check if results are consistent across different runs
4. **Business metrics:** Measure impact on actual security outcomes

Study Resources

Required Reading:

- Scikit-learn documentation on anomaly detection
- Class slides and notebook examples

Recommended Additional Resources:

- **Book:** "Hands-On Machine Learning" by Aurélien Géron (Chapter 9)
- **Paper:** "Anomaly Detection: A Survey" by Chandola et al.
- **Documentation:** [Scikit-learn Anomaly Detection Guide](#)

Practice Datasets:

- Credit card fraud detection (Kaggle)
- Network intrusion detection (KDD Cup 1999)
- NASA bearing failure detection

Technical Setup

Required Software:

Python libraries needed

```
pip install pandas numpy scikit-learn matplotlib seaborn jupyter
```

File Requirements:

- `dataset_malwares.csv` (provided by instructor)
- Jupyter Notebook environment
- Python 3.7 or higher

Common Issues & Solutions:

Problem: ImportError for scikit-learn

Solution: `pip install --upgrade scikit-learn`

Problem: Memory issues with large dataset

Solution: Work with a sample of the data initially: `df.sample(n=5000)`

Problem: Visualization challenges with 69 features

Solution: Use dimensionality reduction (PCA) for visualization

Key Questions for Self-Assessment

1. Can you explain the difference between anomaly detection and supervised classification?
2. What are the trade-offs between the three algorithms we learned?
3. How would you choose the contamination parameter for Isolation Forest?
4. What makes anomaly detection particularly useful for cybersecurity?
5. How can you evaluate anomaly detection when you don't have labels?

Tips for Success

During Class:

- Ask questions when concepts are unclear
- Take notes on parameter meanings and trade-offs
- Try different parameter values during lab time

For the Assignment:

- Start early - technical setup can take time
- Test your code with small data samples first
- Document your LLM prompts and results
- Include explanations of what your code does

For Understanding:

- Practice with different datasets
- Experiment with parameter values
- Think about real-world applications
- Connect concepts to previous classes

Capstone Project Connection:

This assignment builds toward your final capstone project on malware detection using machine learning. The skills you learn here will be essential for:

- Data preprocessing and feature engineering
- Model selection and evaluation
- Understanding unsupervised learning approaches