

Machine Learning Essentials for Cybersecurity

MSCS640: Artificial Intelligence and Cyber Security

Dr. Mouatez Karbab



أكاديمية جوعان بن جاسم للدراسات الدفاعية
Joaan Bin Jassim Academy for Defence Studies

October 2, 2026



Outline

- 1 AI, ML, and DL
- 2 Core ML Paradigms
- 3 Key ML Model Families
- 4 Evaluation Metrics for Security
- 5 Challenges in ML for Cybersecurity
- 6 Hands-on Project



Introduction I

Chapter Objectives

This chapter provides the essential machine learning (ML) knowledge for cybersecurity professionals.

- Distinguish between Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL).
- Explore core paradigms: Supervised, Unsupervised, Semi-supervised, and Reinforcement Learning.
- Learn key model families: from Random Forest to Neural Networks.
- Master practical aspects: Feature Engineering, Evaluation Metrics, and Workflow.
- Address challenges: Imbalanced data, Encryption, and Adversarial attacks.



Outline

- 1 AI, ML, and DL
- 2 Core ML Paradigms
- 3 Key ML Model Families
- 4 Evaluation Metrics for Security
- 5 Challenges in ML for Cybersecurity
- 6 Hands-on Project



AI, ML, and DL Definitions

- **Artificial Intelligence (AI):** The broadest field. Creating machines that perform tasks requiring human intelligence (e.g., decision-making, visual perception).
- **Machine Learning (ML):** A subfield of AI. Gives computers the ability to *learn from data without being explicitly programmed*. Algorithms find patterns to make predictions.
- **Deep Learning (DL):** A specialized subfield of ML. Uses *artificial neural networks* with many layers ("deep") to find complex patterns in unstructured data (images, text).

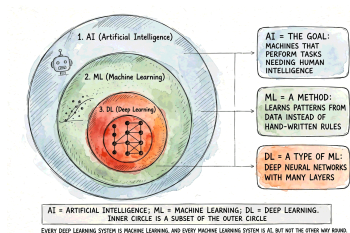


Figure: The hierarchical relationship: DL is part of ML, and ML is part of AI.



Section 1: Assessment (1/3)

Question 1

Which of the following best describes the relationship between AI, ML, and DL?

- A. They are three separate, unrelated fields.
- B. AI is a subset of Machine Learning.
- C. Deep Learning is a subset of Machine Learning, which is a subset of AI.
- D. Machine Learning uses Neural Networks, while AI uses Logic.



Section 1: Assessment (1/3) - Answer

Question 1

Which of the following best describes the relationship between AI, ML, and DL?

- A. They are three separate, unrelated fields.
- B. AI is a subset of Machine Learning.
- C. Deep Learning is a subset of Machine Learning, which is a subset of AI. (✓)
- D. Machine Learning uses Neural Networks, while AI uses Logic.

Explanation

As shown in the Venn diagram, AI is the broad umbrella. ML is the method of learning from data within AI. DL is a specialized technique using neural networks within ML.



Section 1: Assessment (2/3)

Question 2

What is the core characteristic that defines Machine Learning?

- A. It requires explicit programming for every scenario.
- B. It learns patterns from data to make predictions on unseen data.
- C. It only works with images and audio.
- D. It simulates the human brain using neurons.



Section 1: Assessment (2/3) - Answer

Question 2

What is the core characteristic that defines Machine Learning?

- A. It requires explicit programming for every scenario.
- B. It learns patterns from data to make predictions on unseen data. (✓)
- C. It only works with images and audio.
- D. It simulates the human brain using neurons.

Explanation

ML is defined by its ability to learn rules from data rather than having rules hard-coded by a programmer.



Section 1: Assessment (3/3)

Question 3

Deep Learning is particularly effective for which type of data?

- A. Simple spreadsheets.
- B. Unstructured data like images, audio, and text.
- C. Small datasets with few features.
- D. Linear regression problems.



Section 1: Assessment (3/3) - Answer

Question 3

Deep Learning is particularly effective for which type of data?

- A. Simple spreadsheets.
- B. Unstructured data like images, audio, and text. (✓)
- C. Small datasets with few features.
- D. Linear regression problems.

Explanation

The multi-layered structure of Deep Learning networks allows them to automatically extract features from complex, high-dimensional, and unstructured data.



Outline

- 1 AI, ML, and DL
- 2 Core ML Paradigms
- 3 Key ML Model Families
- 4 Evaluation Metrics for Security
- 5 Challenges in ML for Cybersecurity
- 6 Hands-on Project



Supervised vs. Unsupervised Learning

Supervised Learning (By Example)

- **Concept:** Teaching with labels ("This is malware", "This is safe").
- **Best For:** Malware detection and classifying known attacks.

Unsupervised Learning (By Observation)

- **Concept:** Feeding raw data without labels. The AI groups similarities and highlights outliers.
- **Best For:** Insider threats and hunting for new, unknown attacks.



The Data Preparation Pipeline I

"Garbage in, garbage out." Data must be cleaned before training.

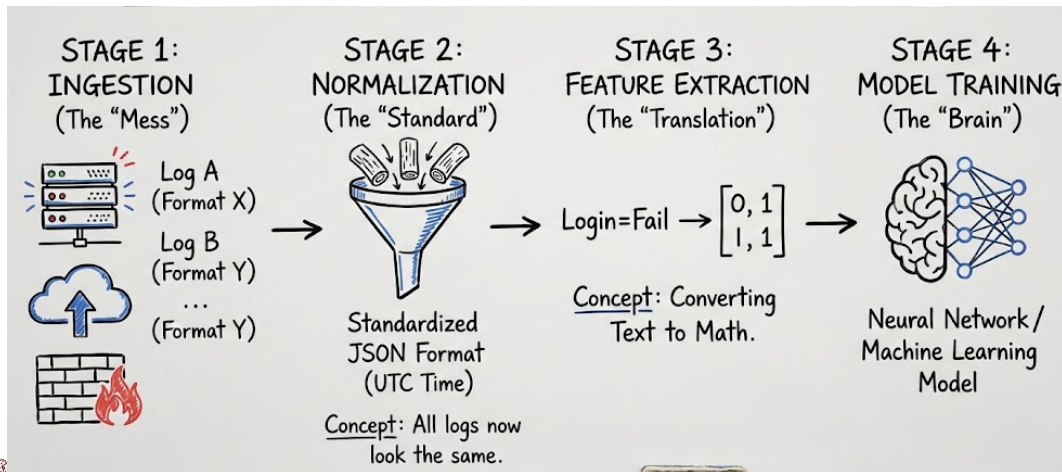
- 1 **Normalization:** Translate the different "languages" of tools (Microsoft vs. Cisco) into one standard format (e.g., JSON).
 - **Tool A says:** "User: Admin logged in."
 - **Tool B says:** "Account: Admin, Action: Authentication Success."
- 2 **Time Synchronization:** Every device must use UTC. If clocks disagree, the cause-and-effect timeline of an attack is destroyed.

Feature Extraction

AI models (Neural Networks) cannot read text. They understand math.

- Raw: "Status=Fail" → Feature: `is_failure` = 1
- Raw: "10:00 AM" → Feature: `hour` = 10

The Data Preparation Pipeline II



Python: Simulating the Data (1/2)

This block simulates loading raw log data. In a real scenario, this would be a large CSV file.

```
import pandas as pd
import io

# 1. Simulate raw log data (CSV format)
log_data = """timestamp,source_ip,status_code
2025-08-16 10:00:01,192.168.1.10,200
2025-08-16 10:00:05,192.168.1.10,404
2025-08-16 10:00:10,192.168.1.10,404
2025-08-16 10:00:15,10.0.0.5,200
"""

# 2. Load the data into a DataFrame
df = pd.read_csv(io.StringIO(log_data))
```

Python: Feature Extraction Logic (2/2)

Here we turn raw "404" codes into a mathematical count. The AI can learn that a high count is suspicious.

```
# 3. FEATURE EXTRACTION
```

```
# Logic: Group by IP address, count how many '404' errors appear.
```

```
features = df.groupby('source_ip')['status_code'].apply(
    lambda x: (x == 404).sum()
).reset_index(name='error_count_feature')
```

```
print("Data ready for AI Model:")
print(features)
```

```
# Output:
```

```
#      source_ip  error_count_feature
# 0      10.0.0.5                    0
# 1  192.168.1.10                    2
```

Supervised Learning Overview

The algorithm learns from a dataset where inputs are paired with correct output labels.

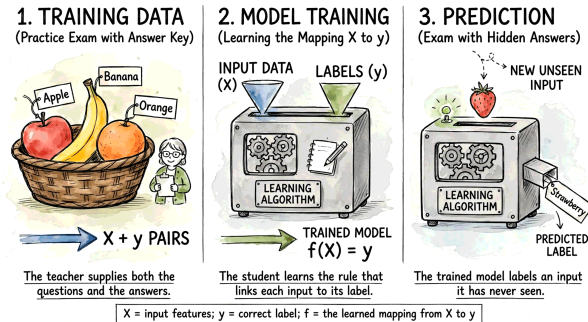


Figure: Supervised Machine Learning: Learning the mapping from Input (X) to Output (y).



Supervised Learning System

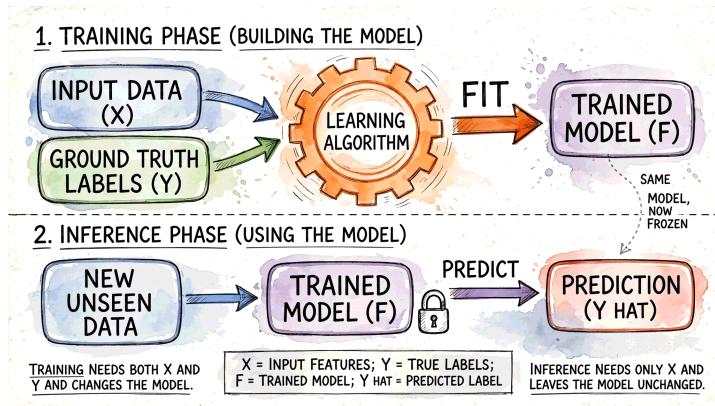


Figure: Supervised Machine Learning System Overview.



Application: Classification

Predicting a discrete category (e.g., Benign vs. Malware).

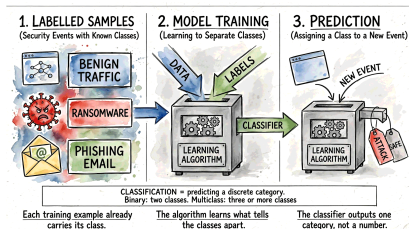


Figure: ML Classification Overview.

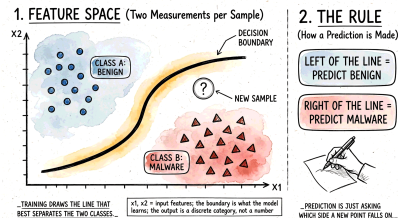


Figure: Classification Boundary.



Code: Supervised Classification (Phishing)

A simple Phishing detector using Logistic Regression. Features: URL length and special character counts.

```

1 import numpy as np
2 from sklearn.linear_model import LogisticRegression
3
4 # --- CLASSIFICATION EXAMPLE: Phishing Detection ---
5 # 1. Dataset (Features: [URL Length, Number of '@' symbols])
6 X_train = np.array([
7     [15, 0], [22, 0], [18, 0], # Benign samples
8     [85, 1], [90, 2], [75, 1] # Phishing samples
9 ])
10 # Labels (0: Benign, 1: Phishing)
11 y_train = np.array([0, 0, 0, 1, 1, 1])
12
13 # 2. Train Model
14 clf = LogisticRegression()
15 clf.fit(X_train, y_train)
16
17 # 3. Predict on new data
18 new_email = np.array([[80, 1]]) # A long URL with an '@' symbol
19 prediction = clf.predict(new_email)
20 print(f"Classification Prediction: {prediction[0]} (1=Phishing, 0=Benign)")

```



Application: Regression

Predicting a continuous numerical value (e.g., Risk Score).

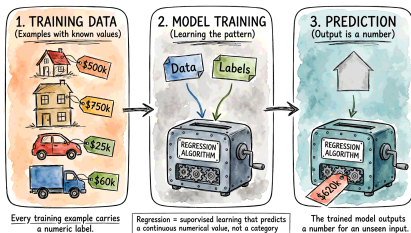


Figure: Regression Overview.

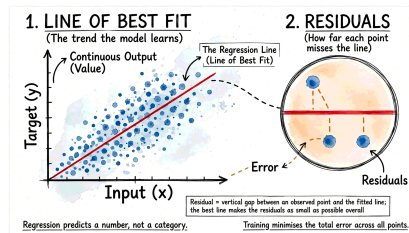


Figure: Fitting a line to predict trends.



Code: Supervised Regression (Risk Score)

Predicting a risk score based on the count of open critical vulnerabilities.

```
1 import numpy as np
2 from sklearn.linear_model import LinearRegression
3
4 # --- REGRESSION EXAMPLE: Risk Scoring ---
5 # 1. Dataset (Feature: [Count of Open Critical Vulnerabilities])
6 X_history = np.array([ [0], [1], [2], [5], [10] ])
7
8 # Labels (Target Risk Score: 0.0 to 100.0)
9 y_history = np.array([10.0, 25.0, 40.0, 65.0, 95.0])
10
11 # 2. Train Model
12 reg = LinearRegression()
13 reg.fit(X_history, y_history)
14
15 # 3. Predict Risk for a system with 7 open vulnerabilities
16 new_system = np.array([[7]])
17 predicted_score = reg.predict(new_system)
18
19 print(f"Regression Prediction (Risk Score): {predicted_score[0]:.2f}")
```



Unsupervised Learning Overview

Working with data that has no predefined labels. The model must find structure on its own.

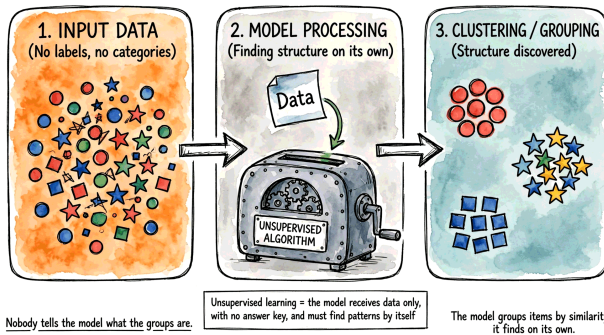


Figure: Unsupervised Learning: Finding structure in unlabeled data.

Application: Anomaly Detection

Identifying outliers significantly different from the norm (e.g., Insider Threats).

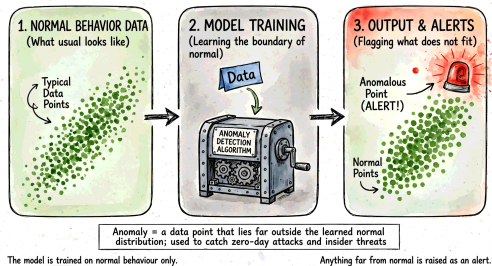


Figure: Anomaly Detection.



Code: Anomaly Detection

Using *Isolation Forest* to detect abnormal file access patterns (Insider Threat).

```

1 from sklearn.ensemble import IsolationForest
2 import numpy as np
3
4 # --- ANOMALY DETECTION: Insider Threat ---
5 # 1. Training Data (Normal User Behavior: Files per minute)
6 X_normal = np.array([[5], [6], [5], [7], [10], [8], [5]])
7
8 # 2. Train Model (contamination=0.01 tells model anomalies are rare)
9 iso = IsolationForest(contamination=0.01, random_state=42)
10 iso.fit(X_normal)
11
12 # 3. Live Data Monitoring
13 # User A (Normal), User B (Data Exfiltration)
14 X_live = np.array([[6], [1000]])
15
16 predictions = iso.predict(X_live)
17 # Output: 1 = Normal, -1 = Anomaly
18 print(f"Predictions: {predictions}")

```



Application: Clustering

Grouping similar data points (e.g., Categorizing malware families).

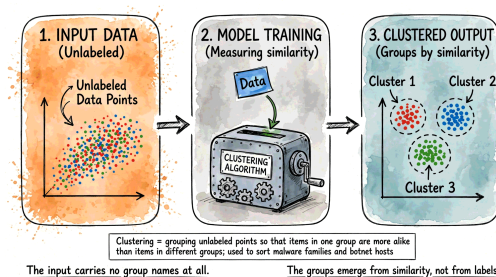


Figure: Clustering: Grouping unlabelled data.



Code: Clustering (K-Means)

Using K-Means to separate DNS traffic from HTTPS traffic without labels.

```

1 from sklearn.cluster import KMeans
2 import numpy as np
3
4 # --- CLUSTERING: Traffic Categorization ---
5 # 1. Unlabeled Data (Features: [Packet Size, Destination Port])
6 X_traffic = np.array([
7     [64, 53], [70, 53], [60, 53],    # Small packets on Port 53 (DNS)
8     [1500, 443], [1450, 443], [1500, 443] # Large packets on Port 443 (HTTPS)
9 ])
10
11 # 2. Train Model (Find 2 clusters)
12 kmeans = KMeans(n_clusters=2, random_state=42)
13 kmeans.fit(X_traffic)
14
15 # 3. Analyze Results
16 print(f"Cluster Labels: {kmeans.labels_}")
17 # Expected: [0, 0, 0, 1, 1, 1] - distinct groups found.

```



Application: Dimensionality Reduction

Compressing complex data while retaining information (e.g., Visualization, preprocessing).

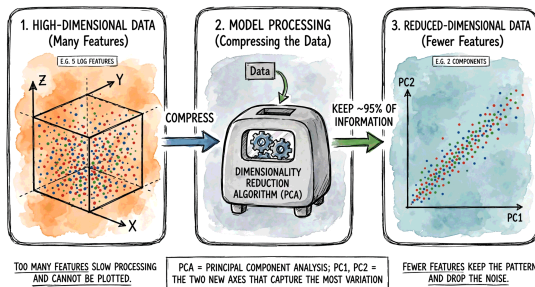


Figure: Dimensionality Reduction.



Code: Dimensionality Reduction (PCA)

Using PCA to compress high-dimensional log data.

```

1 from sklearn.decomposition import PCA
2 import numpy as np
3
4 # --- DIMENSIONALITY REDUCTION: Log Compression ---
5 # 1. High-Dimensional Data (10 samples, 5 features each)
6 X_logs = np.random.rand(10, 5)
7
8 # 2. Apply PCA
9 # Compress 5 features down to 2 "Principal Components"
10 pca = PCA(n_components=2)
11 X_compressed = pca.fit_transform(X_logs)
12
13 # 3. Result
14 print(f"Compressed Shape: {X_compressed.shape}")
15 print(f"Information Retained: {np.sum(pca.explained_variance_ratio_):.2%}")

```



Semi-supervised Learning

Uses a small amount of labeled data and a large amount of unlabeled data.

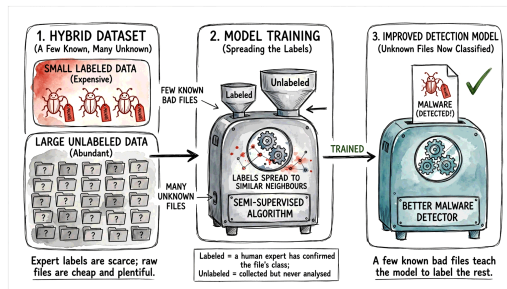


Figure: Propagating labels from known to unknown data.



Code: Semi-supervised Learning

Using Label Spreading to infer malware status for unknown files.

```

1 import numpy as np
2 from sklearn.semi_supervised import LabelSpreading
3
4 # --- SEMI-SUPERVISED EXAMPLE: Malware Label Propagation ---
5 # 1. Dataset (Feature: [File Entropy, File Size])
6 X = np.array([
7     [0.1, 100], [0.2, 120], # Cluster A (Likely Benign)
8     [0.9, 900], [0.8, 950], # Cluster B (Likely Malware)
9     [0.15, 110], [0.85, 920] # New Files (Unknown)
10 ])
11 # 2. Labels (0: Benign, 1: Malware, -1: Unknown)
12 y_mixed = np.array([0, 0, 1, 1, -1, -1])
13
14 # 3. Train Model
15 model = LabelSpreading(kernel='knn', n_neighbors=2)
16 model.fit(X, y_mixed)
17
18 # 4. Results
19 print(f"Inferred Labels: {model.transduction_}")

```



Reinforcement Learning (RL)

An agent learns by trial and error (Actions, Rewards, Penalties).

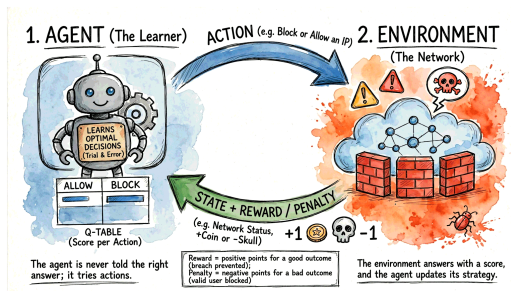


Figure: Reinforcement Learning Loop.



Code: Reinforcement Learning (Firewall Agent)

A simple Q-Learning loop for an autonomous firewall.

```

1 import numpy as np
2 import random
3 # --- RL EXAMPLE: Q-Learning Firewall Agent ---
4 # Actions: 0 = Allow, 1 = Block. State: IP '192.168.1.5'
5 q_table = { '192.168.1.5': [0.0, 0.0] }
6
7 def get_reward(action, is_malicious):
8     if is_malicious and action == 1: return +10 # Good: Blocked attack
9     if is_malicious and action == 0: return -50 # Bad: Allowed attack
10    return +5 if action == 0 else -10 # Valid user logic
11
12 # Simulation Loop
13 ip = '192.168.1.5'; is_malicious = True; learning_rate = 0.5
14 for episode in range(5):
15     # Choose Action (Explore vs Exploit)
16     action = random.choice([0, 1]) if random.random() < 0.5 else np.argmax(q_table[ip])
17
18     reward = get_reward(action, is_malicious)
19
20     # Update Q-Table
21     old_value = q_table[ip][action]
22     q_table[ip][action] = old_value + (learning_rate * reward)
23     print(f"Ep {episode+1}: Action {action} -> Reward {reward}")

```

The Supervised Learning Workflow

A structured approach is critical to prevent data leakage.

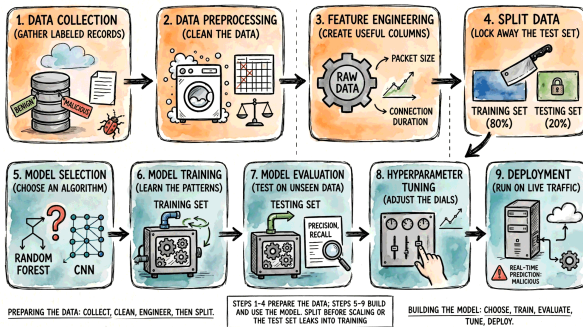


Figure: The Supervised Learning Workflow.

Workflow: 1. Data Collection & 2. Splitting

Split data **immediately** to prevent leakage.

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3
4 # 1. Data Collection (Mock Dataset)
5 data = {
6     'duration': [0.1, 5.2, 0.2, 12.0, 0.1, 0.5, 4.5, 1.2, 0.1, 3.0],
7     'src_bytes': [64, 5000, 128, 9000, 64, 256, 4500, 500, 64, 6000],
8     'dst_bytes': [1000, 0, 500, 0, 800, 1200, 0, 300, 950, 0],
9     'label': [0, 1, 0, 1, 0, 0, 1, 0, 0, 1] # 0: Benign, 1: Malicious
10 }
11 df = pd.DataFrame(data)
12
13 # 2. Data Splitting
14 X = df.drop('label', axis=1); y = df['label']
15
16 # stratify=y ensures both sets maintain the same ratio of attacks
17 X_train, X_test, y_train, y_test = train_test_split(
18     X, y, test_size=0.2, random_state=42, stratify=y
19 )

```



Workflow: 3. Engineering & 4. Preprocessing

Fit scalers ONLY on training data.

```

1 from sklearn.preprocessing import StandardScaler
2
3 # 3. Feature Engineering (Apply to both)
4 X_train['bytes_per_sec'] = X_train['src_bytes'] / (X_train['duration'] + 0.01)
5 X_test['bytes_per_sec'] = X_test['src_bytes'] / (X_test['duration'] + 0.01)
6
7 # 4. Data Preprocessing (Scaling)
8 scaler = StandardScaler()
9
10 # CRITICAL: Fit scaler ONLY on Training data
11 X_train_scaled = scaler.fit_transform(X_train)
12
13 # Transform Test data using learned parameters
14 X_test_scaled = scaler.transform(X_test)

```



Workflow: 5-7. Selection, Tuning, Training

Use Cross-Validation on the training set.

```
1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.model_selection import GridSearchCV
3
4 # 5. Model Selection
5 rf = RandomForestClassifier(random_state=42)
6
7 # 6. Hyperparameter Tuning (Cross-Validation)
8 param_grid = {'n_estimators': [10, 50], 'max_depth': [3, 5]}
9 grid_search = GridSearchCV(rf, param_grid, cv=2, scoring='f1')
10 grid_search.fit(X_train_scaled, y_train)
11
12 # 7. Model Training (Automatic via GridSearchCV)
13 best_model = grid_search.best_estimator_
14 print(f"Best Parameters: {grid_search.best_params_}")
```



Workflow: 8. Evaluation & 9. Deployment

Evaluate on the locked-away test set.

```

1 from sklearn.metrics import classification_report
2
3 # 8. Model Evaluation
4 y_pred = best_model.predict(X_test_scaled)
5 print(classification_report(y_test, y_pred, target_names=['Benign', 'Malicious']))
6
7 # 9. Deployment (Simulation)
8 def analyze_packet(duration, src_bytes, dst_bytes):
9     live_data = pd.DataFrame([[duration, src_bytes, dst_bytes]],
10                               columns=['duration', 'src_bytes', 'dst_bytes'])
11     live_data['bytes_per_sec'] = live_data['src_bytes'] / (live_data['duration'] + 0.01)
12     live_scaled = scaler.transform(live_data)
13     return "ALERT" if best_model.predict(live_scaled)[0] == 1 else "Benign"
14
15 print(analyze_packet(15.0, 10000, 0))

```



Section 2: Assessment (1/3)

Question 1

Which ML paradigm requires a dataset with labeled examples (Input/Output pairs)?

- A. Unsupervised Learning.
- B. Reinforcement Learning.
- C. Supervised Learning.
- D. Clustering.



Section 2: Assessment (1/3) - Answer

Question 1

Which ML paradigm requires a dataset with labeled examples (Input/Output pairs)?

- A. Unsupervised Learning.
- B. Reinforcement Learning.
- C. Supervised Learning. (✓)
- D. Clustering.

Explanation

Supervised learning works like a teacher/student model, requiring a "key" (labels) to learn the relationship between inputs and outputs.



Section 2: Assessment (2/3)

Question 2

What is the primary use case for Unsupervised Learning in cybersecurity?

- A. Phishing detection using known bad URLs.
- B. Anomaly detection (finding zero-day attacks or outliers).
- C. Predicting exact risk scores.
- D. Training a firewall agent via rewards.



Section 2: Assessment (2/3) - Answer

Question 2

What is the primary use case for Unsupervised Learning in cybersecurity?

- A. Phishing detection using known bad URLs.
- B. Anomaly detection (finding zero-day attacks or outliers). (✓)
- C. Predicting exact risk scores.
- D. Training a firewall agent via rewards.

Explanation

Since unsupervised learning doesn't need labels, it is ideal for finding data that deviates from the "norm" without knowing what the attack looks like beforehand.



Section 2: Assessment (3/3)

Question 3

In the Supervised Learning Workflow, why must scaling parameters be calculated only on the Training set?

- A. To save computation time.
- B. To prevent Data Leakage (peeking at the test set).
- C. Because the Test set contains no numbers.
- D. It doesn't matter; you can use the whole dataset.



Section 2: Assessment (3/3) - Answer

Question 3

In the Supervised Learning Workflow, why must scaling parameters be calculated only on the Training set?

- A. To save computation time.
- B. To prevent Data Leakage (peeking at the test set). (✓)
- C. Because the Test set contains no numbers.
- D. It doesn't matter; you can use the whole dataset.

Explanation

The test set must simulate unseen data. Using it to calculate mean/variance for scaling allows the model to "peek" at the distribution of the test data, invalidating the evaluation.



- 


Logistic Regression

A classification algorithm estimating probabilities using a sigmoid function.

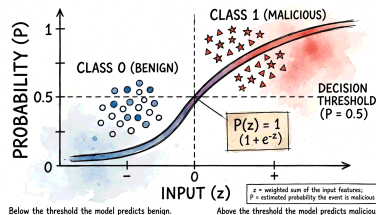


Figure: Logistic Regression Sigmoid Curve.

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train) # Train
```

Support Vector Machines (SVM)

Finds a hyperplane that maximizes the margin between classes.

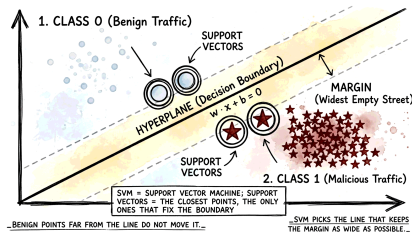
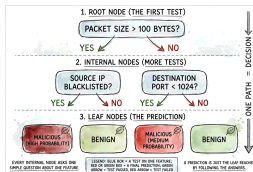


Figure: SVM maximizing the margin.

```
from sklearn.svm import SVC
clf = SVC(kernel='linear', C=1.0)
clf.fit(X_train, y_train)
```

Decision Trees & Ensembles

Decision Trees: Flowchart-like structures. Prone to overfitting.



Ensembles: Combine models (Random Forest, XGBoost) for stability.

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)
```



Neural Networks (NNs)

Layers of interconnected neurons learning weights to minimize error.

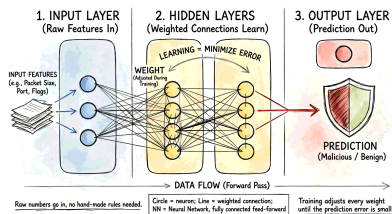


Figure: Feed-Forward Neural Network.

```
import torch.nn as nn
# Input: 10 features -> Hidden: 64 neurons -> Output: 2 classes
model = nn.Sequential(nn.Linear(10, 64), nn.ReLU(), nn.Linear(64, 2))
```



Convolutional Neural Networks (CNNs)

Uses filters [2] to scan for local patterns (e.g., in malware images or byte sequences).

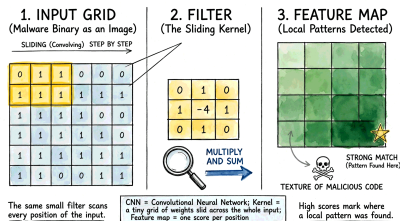
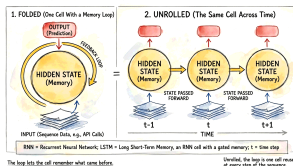


Figure: CNN Filters.

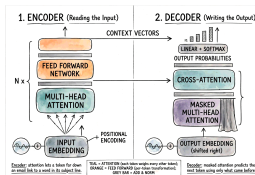
```
# 1D Conv Layer for sequence data (e.g., packet bytes)
conv_layer = nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3)
```

RNNs & Transformers

RNNs/LSTMs [1]: Process sequences with memory (Time-series, API calls).



Transformers [3]: Use "Attention" to link distant parts of data (e.g., Phishing emails).



Feature Engineering: Text (NLP)

Converting text to numbers. Example: Bag-of-Words for Phishing.

```

1 from sklearn.feature_extraction.text import CountVectorizer
2 import pandas as pd
3
4 # 1. Raw Data (Email Subjects)
5 emails = ["Urgent: Update password", "Meeting agenda", "Win free iPhone"]
6
7 # 2. Transform to Numbers
8 vectorizer = CountVectorizer(stop_words='english')
9 X = vectorizer.fit_transform(emails)
10
11 # 3. Result
12 print(pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out()))
13 # Columns: 'urgent', 'win', 'meeting'... Values: 1 or 0

```



Feature Engineering: Temporal & Entropy

Temporal: Extracting 'Hour of Day'. **Entropy:** Measuring randomness (Malware vs. Legit).

```

1 # Temporal Example
2 df['timestamp'] = pd.to_datetime(df['timestamp'])
3 df['hour'] = df['timestamp'].dt.hour # Attackers often work at night
4
5 # Entropy Example (Malware Detection)
6 from scipy.stats import entropy
7 def calc_entropy(data):
8     probs = [count / len(data) for count in collections.Counter(data).values()]
9     return entropy(probs, base=2)
10 # High Entropy = Encrypted/Packed Malware
    
```



Data Normalization

Scaling features to a common range (0-1) so large numbers don't dominate.

```

1 from sklearn.preprocessing import MinMaxScaler
2 import pandas as pd
3
4 # Unscaled: 'Bytes' (10000) dominates 'Hour' (23)
5 df = pd.DataFrame({'bytes': [10000], 'hour': [23]})
6
7 # Apply Scaling
8 scaler = MinMaxScaler()
9 df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
10 # Result: Both features are now between 0.0 and 1.0
  
```



Representation Learning

Deep Learning automates feature extraction using Embeddings.

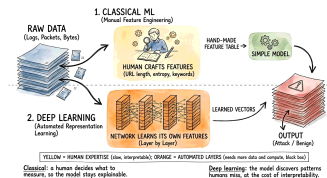


Figure: Manual vs. Automated Feature Learning.

```
1 import torch.nn as nn
2 # Learn a vector of size 3 for every API call automatically
3 embedding_layer = nn.Embedding(num_embeddings=10, embedding_dim=3)
4 learned_features = embedding_layer(raw_sequence)
```



Section 3: Assessment (1/3)

Question 1

Which classical algorithm uses a "sigmoid" function to output a probability between 0 and 1?

- A. Decision Tree.
- B. Support Vector Machine (SVM).
- C. Logistic Regression.
- D. K-Means Clustering.



Section 3: Assessment (1/3) - Answer

Question 1

Which classical algorithm uses a "sigmoid" function to output a probability between 0 and 1?

- A. Decision Tree.
- B. Support Vector Machine (SVM).
- C. Logistic Regression. (✓)
- D. K-Means Clustering.

Explanation

Despite its name, Logistic Regression is a classifier that fits data to a sigmoid (S-shaped) curve, producing a probability output suited for binary classification.



Section 3: Assessment (2/3)

Question 2

What is the primary advantage of a Random Forest over a single Decision Tree?

- A. It is faster to train.
- B. It is easier to interpret visually.
- C. It reduces overfitting and improves stability.
- D. It does not require labeled data.



Section 3: Assessment (2/3) - Answer

Question 2

What is the primary advantage of a Random Forest over a single Decision Tree?

- A. It is faster to train.
- B. It is easier to interpret visually.
- C. It reduces overfitting and improves stability. (✓)
- D. It does not require labeled data.

Explanation

Single decision trees often memorize the training data (overfitting). Random Forest averages the predictions of many trees, smoothing out errors and improving generalization.



Section 3: Assessment (3/3)

Question 3

What is the purpose of Data Normalization (Scaling)?

- A. To convert text into numbers.
- B. To ensure features with large values (e.g., Bytes) don't dominate features with small values (e.g., Hour).
- C. To remove null values from the dataset.
- D. To encrypt the data for privacy.



Section 3: Assessment (3/3) - Answer

Question 3

What is the purpose of Data Normalization (Scaling)?

- A. To convert text into numbers.
- B. To ensure features with large values (e.g., Bytes) don't dominate features with small values (e.g., Hour). (✓)
- C. To remove null values from the dataset.
- D. To encrypt the data for privacy.

Explanation

ML algorithms often rely on distance or magnitude. Without scaling, a feature ranging from 0-10000 will mathematically overpower a feature ranging from 0-1, leading to poor model performance.



- 

Classification Metrics

Accuracy is misleading in security due to the Base Rate Fallacy (imbalanced data).

	Real: Attack	Real: Benign
Pred: Attack	TP	FP
Pred: Benign	FN	TN

Table: Confusion Matrix

- **Precision:** $TP / (TP + FP)$ (Trustworthiness of alerts).
- **Recall:** $TP / (TP + FN)$ (Ability to catch attacks).
- **F1-Score:** Harmonic mean of Precision and Recall.



Code: Evaluating Classifiers

```

1 from sklearn.metrics import classification_report, confusion_matrix
2 import numpy as np
3
4 # 1. Ground Truth (Most are Benign/0)
5 y_true = np.array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1])
6
7 # 2. Model Predictions
8 y_pred = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 0])
9
10 # 3. Evaluation
11 print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
12 print(classification_report(y_true, y_pred, target_names=['Benign', 'Attack']))
13 # Precision (Attack) = 0.67 (2 real out of 3 alerts)
14 # Recall (Attack) = 0.67 (Caught 2 out of 3 attacks)

```



Regression & Clustering Metrics

Regression: MSE punishes large errors more than MAE.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_true, y_pred) # High if outliers exist
```

Clustering: Silhouette Score measures cluster separation (-1 to 1).

```
from sklearn.metrics import silhouette_score
score = silhouette_score(X_features, labels) # Closer to 1 is better
```



Threshold Tuning

Adjusting the decision boundary impacts Precision and Recall.

- **Low Threshold (>0.2):** High Recall (Paranoid/Threat Hunting).
- **High Threshold (>0.9):** High Precision (Automated Blocking).

```

1 # Scenario A: Paranoid Mode (Catch everything)
2 y_pred_paranoid = (y_proba > 0.3).astype(int)
3
4 # Scenario B: Conservative Mode (Avoid false alarms)
5 y_pred_safe = (y_proba > 0.8).astype(int)

```



Section 4: Assessment (1/3)

Question 1

In a dataset with 99 benign packets and 1 malicious packet, a model predicts "Benign" for everything. What is the problem?

- A. The accuracy is 0%.
- B. The accuracy is 99%, but the model missed the attack (Base Rate Fallacy).
- C. The model is overfitting.
- D. The Precision is 100%.



Section 4: Assessment (1/3) - Answer

Question 1

In a dataset with 99 benign packets and 1 malicious packet, a model predicts "Benign" for everything. What is the problem?

- A. The accuracy is 0%.
- B. The accuracy is 99%, but the model missed the attack (Base Rate Fallacy). (✓)
- C. The model is overfitting.
- D. The Precision is 100%.

Explanation

High accuracy is meaningless in imbalanced security data if the minority class (attacks) is completely ignored. Recall would be 0%.



Section 4: Assessment (2/3)

Question 2

Which metric represents the percentage of actual attacks that were successfully detected?

- A. Precision.
- B. Recall.
- C. Accuracy.
- D. Mean Squared Error.



Section 4: Assessment (2/3) - Answer

Question 2

Which metric represents the percentage of actual attacks that were successfully detected?

- A. Precision.
- B. Recall. (✓)
- C. Accuracy.
- D. Mean Squared Error.

Explanation

Recall (or Sensitivity) measures the fraction of relevant instances (attacks) that were retrieved by the model.

Section 4: Assessment (3/3)

Question 3

If you lower the decision threshold for a classifier (e.g., from 0.5 to 0.2), what usually happens?

- A. Recall increases, but False Positives also increase.
- B. Precision increases, and Recall increases.
- C. Recall decreases, and False Positives decrease.
- D. The model stops working.



Section 4: Assessment (3/3) - Answer

Question 3

If you lower the decision threshold for a classifier (e.g., from 0.5 to 0.2), what usually happens?

- A. Recall increases, but False Positives also increase. (✓)
- B. Precision increases, and Recall increases.
- C. Recall decreases, and False Positives decrease.
- D. The model stops working.

Explanation

Lowering the threshold makes the model more "paranoid." It catches more attacks (higher Recall) but also flags more benign items as malicious (more False Positives).



أكاديمية جوعان بن جاسم للدراسات الدفاعية

- 

69 / 92

Encryption, Drift, and Adversarial Attacks I

- **Encryption ("Going Dark"):** 90% of traffic is encrypted.
- *Possible Solution:* Encrypted Traffic Analysis (ETA) using metadata (timing, packet sizes) instead of payload.
 - Sequence of packet lengths.
 - Inter-arrival times (the pause between packets).
 - Handshake patterns (Cipher suites offered).



Encryption, Drift, and Adversarial Attacks II

- **Concept Drift:** Attack patterns change over time (e.g., new ransomware variants).

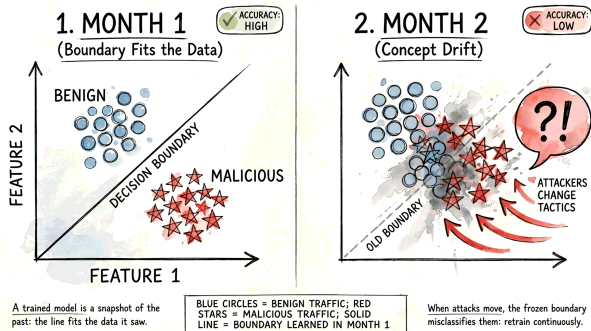


Figure: Concept Drift, As attackers change tactics, the old decision boundary becomes obsolete.



Encryption, Drift, and Adversarial Attacks II

- *Possible Solution:* Continuous Retraining.



Encryption, Drift, and Adversarial Attacks IV

- **Adversarial Examples:** Adding noise to fool the model.

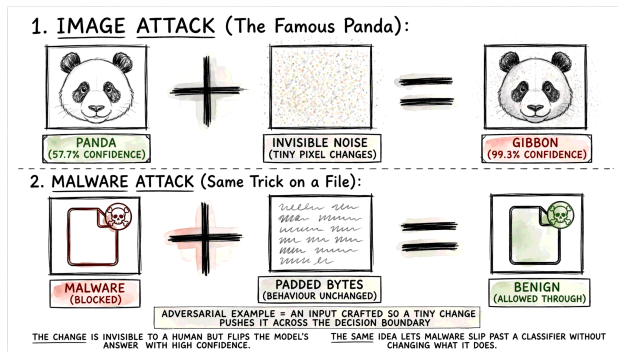


Figure: Adversarial Attacks.



Explainable AI (XAI)

Deep Learning models are "black boxes." If a firewall blocks the CEO's laptop saying "99% Malicious," the security analyst needs to know *why* to verify the alert.

Using SHAP to explain "Black Box" models.

```
1 import shap
2 # Use SHAP to explain a single prediction
3 explainer = shap.Explainer(model)
4 shap_values = explainer(X_test)
5
6 # Visualization tells analyst which feature caused the alert
7 shap.plots.waterfall(shap_values[0])
```



Section 5: Assessment (1/3)

Question 1

What is SMOTE used for?

- A. Encrypting data.
- B. Explaining model predictions.
- C. Balancing datasets by generating synthetic minority samples.
- D. Detecting concept drift.



Section 5: Assessment (1/3) - Answer

Question 1

What is SMOTE used for?

- A. Encrypting data.
- B. Explaining model predictions.
- C. Balancing datasets by generating synthetic minority samples. (✓)
- D. Detecting concept drift.

Explanation

SMOTE (Synthetic Minority Over-sampling Technique) creates fake but realistic examples of the minority class (e.g., attacks) to prevent the model from ignoring them.



Section 5: Assessment (2/3)

Question 2

How can ML models detect attacks in encrypted traffic (HTTPS) if they cannot read the payload?

- A. They break the encryption using brute force.
- B. They analyze metadata like packet timing, sizes, and handshake patterns.
- C. They cannot; ML is useless against encryption.
- D. They ask the user for the password.



Section 5: Assessment (2/3) - Answer

Question 2

How can ML models detect attacks in encrypted traffic (HTTPS) if they cannot read the payload?

- A. They break the encryption using brute force.
- B. They analyze metadata like packet timing, sizes, and handshake patterns. (✓)
- C. They cannot; ML is useless against encryption.
- D. They ask the user for the password.

Explanation

Encrypted Traffic Analysis (ETA) relies on side-channel information (metadata) which remains visible even when the content is encrypted.

Section 5: Assessment (3/3)

Question 3

What is an Adversarial Example?

- A. A software bug in the ML library.
- B. An input with carefully crafted noise designed to fool the model.
- C. A sample of data that was labeled incorrectly by a human.
- D. A model that competes against another model.



Section 5: Assessment (3/3) - Answer

Question 3

What is an Adversarial Example?

- A. A software bug in the ML library.
- B. An input with carefully crafted noise designed to fool the model. (✓)
- C. A sample of data that was labeled incorrectly by a human.
- D. A model that competes against another model.

Explanation

Adversarial examples exploit the mathematical boundaries of a model, using invisible perturbations to force a misclassification (e.g., making malware look benign).



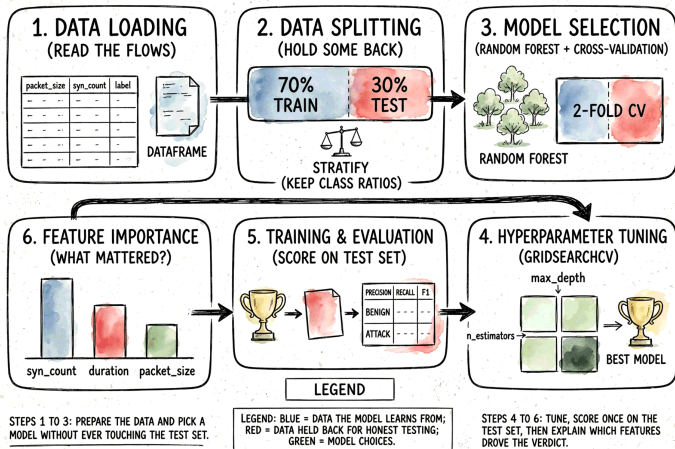
Outline

- 1 AI, ML, and DL
- 2 Core ML Paradigms
- 3 Key ML Model Families
- 4 Evaluation Metrics for Security
- 5 Challenges in ML for Cybersecurity
- 6 Hands-on Project



Project: Threat Classification Workflow

We will build a simple Network Intrusion Detection System (NIDS) using Random Forest.



Project Code: Loading & Splitting

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3
4 # 1. Create Data (Features: Packet Size, SYN Count, Duration)
5 data = {
6     'packet_size': [64, 1500, 64, 200, 64, 1500, 500, 64, 1500, 64],
7     'syn_count': [0, 0, 1, 50, 0, 0, 20, 0, 0, 100],
8     'duration': [0.1, 5.2, 0.1, 1.5, 0.2, 10.0, 0.5, 0.1, 8.5, 2.0],
9     'label': [0, 0, 0, 1, 0, 0, 1, 0, 0, 1] # 1 = Attack
10 }
11 X = pd.DataFrame(data).drop('label', axis=1)
12 y = pd.DataFrame(data)['label']
13
14 # 2. Split (Stratify ensures attack ratio is preserved)
15 X_train, X_test, y_train, y_test = train_test_split(
16     X, y, test_size=0.3, random_state=42, stratify=y
17 )

```



Project Code: Tuning & Training

```

1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.model_selection import GridSearchCV
3
4 # 3 & 4. Grid Search with Cross-Validation
5 param_grid = {'n_estimators': [10, 50], 'max_depth': [None, 5]}
6
7 grid_search = GridSearchCV(
8     RandomForestClassifier(random_state=42),
9     param_grid,
10    cv=2,
11    scoring='f1' # Optimize for F1
12 )
13 grid_search.fit(X_train, y_train)
14 best_model = grid_search.best_estimator_
15
16 print(f"Best Parameters: {grid_search.best_params_}")

```



Project Code: Evaluation & Interpretation

```

1 from sklearn.metrics import classification_report
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # 5. Evaluation
6 y_pred = best_model.predict(X_test)
7 print(classification_report(y_test, y_pred, target_names=['Benign', 'Attack']))
8
9 # 6. Feature Importance
10 importances = best_model.feature_importances_
11 indices = np.argsort(importances)[::-1]
12
13 plt.figure(figsize=(8, 4))
14 plt.bar(range(X.shape[1]), importances[indices])
15 plt.xticks(range(X.shape[1]), X.columns[indices])
16 plt.title("Feature Importance"); plt.show()

```



Conclusion

Summary

- **Hierarchy:** AI > ML > DL.
- **Paradigms:** Supervised (labeled), Unsupervised (unlabeled), RL (rewards).
- **Workflow:** Split data first to avoid leakage. Train, Tune, Evaluate.
- **Metrics:** Use Precision/Recall/F1 for security, not just Accuracy.
- **Challenges:** Must handle imbalanced data, encryption, and adversarial threats.

Next Steps

With this foundation and the hands-on workflow, you are ready to explore advanced defense systems.



References I

- [1] Sepp Hochreiter and Jürgen Schmidhuber.
Long short-term memory.
Neural computation, 9(8):1735–1780, 1997.
- [2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324, 1998.
- [3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
Advances in neural information processing systems, 30, 2017.



Acronyms and Terminology I

AI (Artificial Intelligence) The broad field of computer science focused on creating machines or systems that can perform tasks normally requiring human intelligence.

AUC-ROC (Area Under the ROC Curve) A single value from 0 to 1 that summarizes a model's performance across all classification thresholds. A value closer to 1 indicates a better model.

Adversarial Example An input to an AI model that has been intentionally modified by an attacker to cause the model to make a mistake.

Anomaly Detection A technique, typically from unsupervised learning, used to identify data points or events that are significantly different from the norm. It is crucial for detecting novel threats like zero-day attacks.

Black Box A term for a complex AI model, like a deep neural network, whose internal decision-making process is not easily understood by humans.

CNN (Convolutional Neural Network) A type of deep learning model that excels at finding local patterns in grid-like data, such as images. In cybersecurity, it can analyze malware binaries represented as images.

Classification A supervised learning task where the goal is to predict a discrete category, such as labeling network traffic as "benign" or "attack".



Acronyms and Terminology II

- Clustering** An unsupervised learning technique that groups similar data points together, which can be used to identify new families of malware or related security events.
- Concept Drift** The change in the statistical properties of data over time, which can cause a machine learning model's performance to degrade if it is not retrained.
- Cross-Validation** A method for assessing how the results of a statistical analysis will generalize to an independent dataset. It helps in getting a more reliable estimate of model performance.
- DL (Deep Learning)** A subfield of machine learning that uses multi-layered artificial neural networks (deep neural networks) to learn complex patterns from large amounts of data.
- Ensemble Methods** Techniques that combine the predictions from multiple machine learning models to produce a more accurate and robust prediction than any single model.
- Explainable AI (XAI)** A field of AI research that aims to create systems whose decisions can be easily understood and trusted by humans.
- F1-score** A performance metric that provides a single score by balancing both precision and recall, making it very useful for evaluating models on imbalanced datasets.
- FN (False Negative)** A critical error where a model fails to detect a real attack, classifying it as benign. This can lead to a security breach.



Acronyms and Terminology III

- FP (False Positive)** An error where a model incorrectly flags a benign event as an attack. Too many FPs can lead to alert fatigue for security analysts.
- Feature Engineering** The process of using domain knowledge to select, create, and transform numerical features from raw data to improve a machine learning model's performance.
- Hyperparameters** The configuration settings of a model that are not learned from the data but are set before the training process begins, such as the number of trees in a Random Forest.
- Imbalanced Data** A common scenario in cybersecurity datasets where instances of one class (e.g., malicious events) are far less frequent than another (e.g., benign events).
- LLM (Large Language Model)** A very large deep learning model, typically based on the Transformer architecture, designed to understand and generate human-like text.
- LSTM (Long Short-Term Memory)** An advanced type of Recurrent Neural Network (RNN) that is particularly effective at learning from long sequences of data, such as logs or API call sequences.
- ML (Machine Learning)** A subfield of AI that gives computers the ability to learn from data to make predictions or decisions, without being explicitly programmed for every scenario.
- NLP (Natural Language Processing)** A field of AI focused on enabling computers to understand, interpret, and process human language, used for tasks like analyzing phishing emails.



Acronyms and Terminology IV

Overfitting A problem where a model learns the training data too well, including its noise, which results in poor performance on new, unseen data.

PR (Precision-Recall) Curve A graph that plots the trade-off between a model's precision and recall at various thresholds, especially useful for evaluating models on imbalanced datasets.

Precision An evaluation metric that answers: "Out of all the events the model predicted were malicious, how many actually were malicious?" High precision reduces false alarms.

RL (Reinforcement Learning) A paradigm of machine learning where an agent learns to make optimal decisions by performing actions in an environment and receiving rewards or penalties.

RNN (Recurrent Neural Network) A type of neural network designed for handling sequential data, like a series of log events, because it has an internal memory.

Random Forest A powerful and popular ensemble machine learning model that builds a multitude of decision trees and outputs the class that is the mode of the classes of the individual trees.

Recall Also known as Sensitivity or True Positive Rate. An evaluation metric that answers: "Out of all the actual malicious events, how many did the model correctly identify?" High recall is critical for not missing threats.

Regression A supervised learning task where the goal is to predict a continuous numerical value, such as the severity score of a software vulnerability.

Acronyms and Terminology V

Representation Learning An approach, often associated with deep learning, where the model automatically discovers the most useful features or representations from raw data.

ROC (Receiver Operating Characteristic) Curve A graph that illustrates the performance of a classification model by plotting the true positive rate against the false positive rate at various thresholds.

SVM (Support Vector Machine) A supervised learning model that finds the optimal boundary (hyperplane) to separate data points into different classes.

Semi-supervised Learning A hybrid training approach that uses a combination of a small amount of labeled data and a large amount of unlabeled data.

Supervised Learning The most common machine learning paradigm, where an algorithm learns from a dataset in which each data point is labeled with the correct output.

TN (True Negative) A correct prediction where the model identifies a benign event as benign.

TP (True Positive) A correct prediction where the model identifies a malicious event as malicious.

Testing Set The portion of the dataset that is held back from the training process and is used only at the end to evaluate the final model's performance on unseen data.

Training Set The portion of the dataset used to train a machine learning model. The model learns patterns and relationships from this data.

Unsupervised Learning A machine learning paradigm where the algorithm works with unlabeled data to find hidden patterns or structures on its own.



