

AICS Assignment #6: Model Training & Selection for Malware Detection

AICS Instructor: Steve Smith

Due Date: August 28, 11:59 PM ET

Grade: 18 points

Objective

Train and compare supervised machine learning models using your preprocessed dataset and engineered features from previous assignments. Integrate the unsupervised anomaly detection methods from Assignments #3 and #5 to create a comprehensive comparison of different machine learning approaches. Select the best-performing model for your capstone malware detection project.

Requirements

Data Preparation & Setup (3 points)

- Load your preprocessed dataset from Assignment #2
- Apply engineered features from Assignment #4
- Split data into training (70%) and testing (30%) sets using stratified sampling
- Verify class balance and data shape
- Import required libraries (scikit-learn, pandas, numpy, matplotlib)

Supervised Model Training (6 points)

Train these 4 supervised learning algorithms:

Logistic Regression (1.5 points)

- Train with default parameters first
- Try different regularization strengths: $C = [0.1, 1, 10]$
- Calculate accuracy, precision, recall, and F1-score

Random Forest (1.5 points)

- Train with default parameters first
- Try different numbers of trees: $n_estimators = [50, 100, 200]$
- Analyze feature importance

Support Vector Machine (1.5 points)

- Train with RBF kernel
- Try different C values: [0.1, 1, 10]
- Compare with linear kernel

Gradient Boosting (1.5 points)

- Use HistGradientBoostingClassifier
- Try different learning rates: [0.1, 0.2, 0.3]
- Compare with Random Forest performance

Unsupervised Algorithm Integration (3 points)

Apply the 3 unsupervised algorithms from Assignments #3 and #5:

Isolation Forest (1 point)

- Use your implementation from Assignment #3
- Apply to test set and generate anomaly scores
- Convert anomaly predictions (-1, 1) to binary classification (0, 1)
- Compare performance with supervised models

K-Means Clustering (1 point)

- Use your clustering approach from Assignment #3
- Calculate distances from cluster centers to identify outliers
- Set threshold based on distance percentiles (try 90th, 95th, 99th percentiles)
- Evaluate as binary classifier

One-Class SVM (1 point)

- Use your One-Class SVM from Assignment #3
- Apply with optimized nu parameter from previous work
- Generate decision function scores
- Compare boundary-based detection with supervised approaches

Hybrid Analysis:

- Compare how supervised vs. unsupervised methods perform
- Identify samples where both approaches agree/disagree
- Analyze if combining both improves overall detection

Model Evaluation & Comparison (4 points)

Performance Metrics (2 points) Create a comparison table showing for each model:

- Accuracy score
- Precision (important for reducing false alarms)
- Recall (important for catching malware)
- F1-score

Confusion Matrix Analysis (1 point)

- Generate confusion matrices for all 4 models
- Identify which model has:
 - Fewest false negatives (missed malware)
 - Fewest false positives (false alarms)
 - Best overall balance

ROC Curve Comparison (1 point)

- Plot ROC curves for all models on same graph
- Calculate AUC scores
- Determine which model has best discriminative ability

LLM-Assisted Development (2 points)

Document Your LLM Usage

- Include at least 3 different prompts you used to generate Python code
- Show how you refined prompts to get better results
- Document any debugging assistance received from LLMs
- Demonstrate iterative improvement of your prompts

Example Effective Prompts:

- "Generate Python code to train and compare 4 machine learning models (Logistic Regression, Random Forest, SVM, Gradient Boosting) for binary malware classification. Include accuracy, precision, recall, and F1-score calculation."
- "Create a function to plot ROC curves for multiple trained models on the same graph with AUC scores in the legend using matplotlib."
- "Write code to create a comparison table showing model performance metrics and save the best performing model using joblib."

Model Selection & Justification (2 points)

Final Model Selection

- Choose the best model based on your evaluation
- Provide clear justification considering:
 - Accuracy vs. interpretability trade-offs
 - False positive vs. false negative costs in cybersecurity
 - Computational requirements for deployment

Feature Importance Analysis

- Show top 10 most important features from your best model
- Relate findings back to cybersecurity domain knowledge
- Discuss any surprising or concerning feature dependencies



Deliverables

- Jupyter notebook with all code, outputs, and analysis
- **LLM prompts documentation** in markdown cells (minimum 3 prompts)
- Model comparison table including both supervised and unsupervised methods
- ROC curve plots for all approaches
- Hybrid analysis comparing supervised vs. unsupervised performance
- Written justification for final model selection (2-3 paragraphs)
- Saved final model (using pickle or joblib) ready for capstone deployment



LLM Prompting Guidelines



Effective Prompt Examples:

Model Training Prompt: "Create Python code to train a Random Forest classifier for malware detection using scikit-learn. Include parameter tuning for n_estimators values [50, 100, 200] and calculate accuracy, precision, recall, and F1-score on test data."





Visualization Prompt: "Generate Python code to create a side-by-side comparison of confusion matrices for 4 different machine learning models using matplotlib. Include model names as titles and proper labeling."

Evaluation Prompt: "Write a function that takes multiple trained models (both supervised and unsupervised) and creates a pandas DataFrame comparing their performance metrics (accuracy, precision, recall, F1-score) with the model names as rows. Include conversion logic for unsupervised anomaly scores to binary predictions."

Unsupervised Integration Prompt: "Create Python code to apply Isolation Forest, K-Means clustering, and One-Class SVM from Assignment #3 to a test dataset for malware detection. Convert their outputs to binary predictions and compare with supervised learning results using scikit-learn metrics."

Debugging Prompt: "I'm getting a 'ValueError: Input contains NaN' error when training my SVM model. Here's my code: [paste code]. How can I identify and handle missing values in my dataset?"

Avoid These Prompt Mistakes:

-  "Write some machine learning code"
-  "Make a model comparison"
-  "Fix this error" (without providing context or code)
-  "Create the best malware detector" (too vague)



Documentation Requirements:

Include in markdown cells before each major code section:

- **Original Prompt:** The exact prompt you used
- **Refinements:** How you improved the prompt if needed
- **Modifications:** Any changes you made to the generated code
- **Learning Notes:** What you learned from the LLM interaction



Code Structure Template

```
# 1. Data Loading and Preparation

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, IsolationForest

from sklearn.svm import SVC, OneClassSVM

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc
```

```
import numpy as np

# Load your preprocessed data

X = # your features

y = # your target variable

# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

                                                    stratify=y,

                                                    random_state=42)

# 2. Train Supervised Models

supervised_models = {

    'Logistic Regression': LogisticRegression(),

    'Random Forest': RandomForestClassifier(),

    'SVM': SVC(probability=True),

    'Gradient Boosting': GradientBoostingClassifier()

}

supervised_results = {}

for name, model in supervised_models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    supervised_results[name] = {'accuracy': accuracy, 'model': model}

# 3. Apply Unsupervised Models (from Assignment #3)
```

```

# Isolation Forest

iso_forest = IsolationForest(contamination=0.1, random_state=42)

iso_forest.fit(X_train[y_train == 0]) # Train on benign samples only

iso_pred = iso_forest.predict(X_test)

iso_pred_binary = np.where(iso_pred == -1, 1, 0) # Convert to binary

# K-Means Clustering

kmeans = KMeans(n_clusters=2, random_state=42)

kmeans.fit(X_train)

# Calculate distances and set threshold for outlier detection

distances = kmeans.transform(X_test)

min_distances = np.min(distances, axis=1)

threshold = np.percentile(min_distances, 95) # 95th percentile

kmeans_pred = (min_distances > threshold).astype(int)

# One-Class SVM

oc_svm = OneClassSVM(nu=0.1)

oc_svm.fit(X_train[y_train == 0]) # Train on benign samples only

oc_pred = oc_svm.predict(X_test)

oc_pred_binary = np.where(oc_pred == -1, 1, 0) # Convert to binary

# 4. Compare All Methods

all_results = {}

# Add supervised results

all_results.update(supervised_results)

```

```

# Add unsupervised results

all_results['Isolation Forest'] = {'accuracy': accuracy_score(y_test,
iso_pred_binary)}

all_results['K-Means Outlier'] = {'accuracy': accuracy_score(y_test,
kmeans_pred)}

all_results['One-Class SVM'] = {'accuracy': accuracy_score(y_test,
oc_pred_binary)}

# 5. Visualize and Compare Results

# Create comparison table

# Plot ROC curves for all methods

# Generate confusion matrices

# 6. Select Best Model and Save

best_model = # your selection logic








import joblib

joblib.dump(best_model, 'final_malware_model.pkl')

```

Learning Outcomes

By completing this assignment, you will:

-  Apply supervised learning to cybersecurity problems
-  Integrate unsupervised anomaly detection with supervised classification
-  Compare different machine learning paradigms systematically
-  Understand evaluation metrics relevant to malware detection
-  Make informed model selection decisions for production deployment
-  Create hybrid detection systems combining multiple approaches
-  Prepare a trained model for your capstone project



Common Pitfalls to Avoid

- ❌ Not using stratified sampling for train/test split
- ❌ Forgetting to set `random_state` for reproducible results
- ❌ Only looking at accuracy (precision/recall are crucial for security)
- ❌ Not considering computational costs for deployment
- ❌ Selecting model based on single metric without context
- ❌ Using vague LLM prompts that generate generic code
- ❌ Not documenting your LLM interactions and learning process



Success Tips

- ✅ Start with default parameters, then tune systematically
- ✅ Focus on metrics most important for malware detection (recall)
- ✅ Consider real-world deployment constraints
- ✅ Document your decision-making process clearly
- ✅ **Use specific, detailed prompts when working with LLMs**
- ✅ **Iterate on prompts to get exactly what you need**
- ✅ **Test and understand all LLM-generated code before using it**
- ✅ Save your final model - you'll need it for the capstone!

Note: This assignment directly prepares your model for Assignment #7 (Capstone Deployment), where you'll create a web application or API using your selected model.