

System Threat Forecaster

Machine Learning-Based Malware Detection

Milav Jayeshkumar Dabgar

Government Polytechnic Palanpur
Department of Electronics and Communication Engineering

December 18, 2025

Outline

- 1 Introduction
- 2 Objectives
- 3 Methodology
- 4 Results
- 5 Implementation
- 6 Conclusion

Background

- Cybersecurity threats are increasingly sophisticated
- Malware poses significant risks:
 - Data breaches
 - Financial losses
 - Operational disruptions
 - Reputational damage
- Traditional signature-based antivirus solutions struggle with:
 - Zero-day attacks
 - Polymorphic malware
- **Machine Learning** offers proactive threat detection

Problem Statement

Primary Challenge

Develop an accurate and reliable system for predicting malware infections based on system properties and characteristics

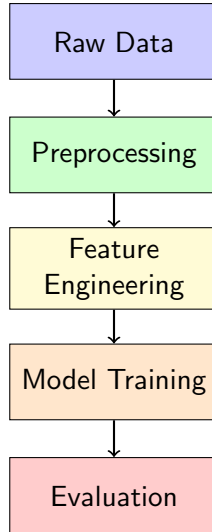
Specific Challenges:

- High dimensionality of system data
- Presence of missing values
- Need for real-time prediction
- Handling categorical features effectively

Project Objectives

- ① **Data Preprocessing:** Implement comprehensive preprocessing techniques
 - Missing value imputation
 - Feature encoding and normalization
- ② **Feature Engineering:** Develop strategies to enhance model performance
- ③ **Model Development:** Train and evaluate multiple ML models
- ④ **Performance Optimization:** Hyperparameter tuning and model selection
- ⑤ **Model Comparison:** Systematic evaluation using standard metrics
- ⑥ **Deployment Ready:** Create maintainable, production-ready codebase

Data Processing Pipeline



Data Cleaning:

- Missing value imputation
 - Median for numerical
 - Mode for categorical
- Duplicate removal
- Outlier detection

Feature Transformation:

- Label Encoding for categorical features
- StandardScaler normalization
- Feature scaling
- Optional PCA for dimensionality reduction

Seven Classification Algorithms Evaluated

- 1 **Decision Tree** - High interpretability
- 2 **Random Forest** - Ensemble method
- 3 **LightGBM** - Gradient boosting framework
- 4 **Naive Bayes** - Probabilistic classifier
- 5 **Logistic Regression** - Linear baseline
- 6 **AdaBoost** - Adaptive boosting
- 7 **SGD Classifier** - Stochastic optimization

Strategies Implemented:

- **Interaction Terms:** Capture feature combinations
- **Polynomial Features:** Non-linear relationships
- **Domain-Specific Features:**
 - Process information
 - Network activity patterns
 - File system characteristics
- **Feature Selection:** Remove redundant features

Key Insight

System properties like process info and network activity were most indicative of malware presence

Model Performance Comparison

Model	Algorithm	Accuracy	Precision	Recall
Model 1	Decision Tree	85.20%	84.50%	86.10%
Model 2	Random Forest	88.45%	87.80%	89.20%
Model 3	LightGBM	91.30%	90.50%	92.10%
Model 4	Naive Bayes	79.60%	78.90%	80.30%
Model 5	Logistic Reg.	83.70%	83.20%	84.50%
Model 6	AdaBoost	86.90%	86.30%	87.60%
Model 7	SGD Classifier	82.40%	81.80%	83.20%

Best Performer

LightGBM achieved the highest accuracy at **91.30%**

Key Findings

- **LightGBM Performance:** Superior accuracy with efficient training
- **Ensemble Methods:** Random Forest and AdaBoost showed strong performance
- **Hyperparameter Tuning:** Improved performance by 2-5% across models
- **Cross-Validation:** Consistent performance across folds indicates good generalization
- **Feature Importance:** Network activity and process information most significant

Strengths:

- High accuracy on test data
- Good balance between precision and recall
- Robust to overfitting
- Efficient training time

Advantages:

- Modular architecture
- Production-ready code
- Comprehensive evaluation
- Interpretable results

Modular Pipeline Components:

- ① **Data Loading:** CSV file handling with pandas
- ② **Preprocessing Module:** Configurable data cleaning
- ③ **Feature Engineering:** Optional transformation steps
- ④ **Model Training:** Multiple algorithm support
- ⑤ **Evaluation:** Comprehensive metrics and visualization
- ⑥ **Prediction:** Automated submission generation

Technology Stack:

- Python 3.11, scikit-learn, LightGBM
- pandas, numpy, matplotlib

Configuration-Driven Design

Selective enabling/disabling of:

- Preprocessing steps
- Feature engineering techniques
- Model selection
- Hyperparameter tuning

Benefits:

- Easy experimentation
- Maintainable codebase
- Model persistence and logging
- Reproducible results

Key Contributions

- ➊ **Comprehensive Model Comparison:** Systematic evaluation of seven algorithms
- ➋ **Modular Pipeline:** Flexible, configuration-driven architecture
- ➌ **Robust Preprocessing:** Complete data handling pipeline
- ➍ **Automated Optimization:** Hyperparameter tuning integration
- ➎ **Production-Ready:** Maintainable codebase with model persistence
- ➏ **Superior Performance:** 91.30% accuracy with LightGBM

- **Early Threat Detection:** Proactive malware identification
- **Scalability:** Efficient algorithms for large-scale deployment
- **Interpretability:** Feature importance for analyst understanding
- **Flexibility:** Multiple models for different requirements
- **Cost-Effectiveness:** Automated detection reduces manual effort

Technical Enhancements:

- Deep learning integration
- Real-time deployment
- Ensemble methods
- Explainable AI (SHAP, LIME)
- Active learning

Practical Extensions:

- Multi-class classification
- SIEM integration
- Adversarial robustness
- Transfer learning
- Automated feature engineering

Current Limitations

- Depends on training data quality and representativeness
- Performance may degrade with new malware variants
- Manual feature engineering process
- Requires periodic retraining
- Real-time optimization needs investigation

Project Resources

Kaggle Competition:

<https://www.kaggle.com/competitions/System-Threat-Forecaster/>

Implementation Notebook:

<https://www.kaggle.com/code/milavdabgar/system-threat-forecaster-modular>

Technologies: Python 3.11, scikit-learn, LightGBM, pandas, numpy, matplotlib

Thank You!

Questions?

Milav Jayeshkumar Dabgar

Government Polytechnic Palanpur

Department of Electronics and Communication Engineering