**Technical Report on Malware Classifier Training**

Jeffrey Jeyachandren, Jia Hua Li, Amber Goel

## Abstract

This report presents the development and evaluation of machine learning classifiers for malware detection based on opcode sequences. Three feature types were analyzed: 1-gram, 2-gram, and a combined feature set, with class balancing techniques (RandomOverSampler and SMOTE) applied to address dataset imbalances. Classifiers tested include Support Vector Machine (SVM), K-Nearest Neighbors (KNN) with k=3 and k=5, and Decision Tree. The results indicate that the KNN classifier with k=3 achieved the highest accuracy (~90%) across all feature types, with the combined features yielding consistent performance improvements for most models. SVM exhibited moderate performance (~70%), while Decision Tree struggled with data sensitivity despite class balancing (~50%). Overall, this study demonstrates the effectiveness of balancing techniques and n-gram feature extraction for improving malware detection, highlighting KNN as the most reliable classifier for this task. Visualizations, including confusion matrices and comparative accuracy plots, provide detailed insights into classifier performance. The opcodes and Jupyter Notebook which contains data processing, result metrics, and data visualizations can be found at https://github.com/JeffinWithYa/apt_analysis_and_samples .

## Implementation Methodology

To prepare the malware opcode data for training and testing the classifier, the following pre-processing steps were performed:

### 1. Data Collection and Organization

- The opcode data was stored in directories organized by Advanced Persistent Threat (APT) groups. Each directory represented a specific APT group and contained text files with opcode sequences extracted from malware samples associated with that group.

### 2. Reading and Structuring the Data

- A script was developed to traverse the directory structure, read each opcode file, and extract the opcode sequences.
- Opcodes within each file were separated by newline characters. These were concatenated into comma-separated strings for uniform representation.
- The extracted opcode sequences were stored in a DataFrame alongside their respective APT group labels. Example:

| | Opcodes | APT |
|---|---|---|
| 0 | JMP, JMP, JMP, JMP, JMP, JMP, JMP, JMP, POP, P... | Evilnum |
| 1 | PUSH, MOV, MOV, CALL, PUSH, PUSH, PUSH, PUSH, ... | APT19_opcodes |
| 2 | MOV, PUSH, MOV, CALL, TEST, PUSH, CALL, ADD, M... | APT19_opcodes |
| 3 | PUSH, PUSH, PUSH, MOV, CMP, JE, CMP, JNE, PUSH... | APT19_opcodes |

- 4 PUSH, PUSH, PUSH, MOV, CMP, JE, CMP, JNE, PUSH... APT19_opcodes

**4. Feature Engineering**

- To represent opcode sequences numerically for machine learning, **n-gram features** were extracted:
  - **1-gram Features**:
    - Unique individual opcodes were identified across the entire dataset.
    - A frequency matrix was created where each row represented a malware sample, and columns represented unique opcodes. Values indicated the frequency of each opcode in a sample.
  - **2-gram Features**:
    - Opcode sequences were analyzed to extract consecutive pairs of opcodes (bigrams).
    - A similar frequency matrix was constructed for bigrams, with each row representing a sample and columns representing unique bigrams.
  - **Combined Features:**
    - The 1-gram and 2-gram feature matrices were concatenated horizontally to create a comprehensive feature set.
    - This combined feature set ensured that both individual opcode frequencies and sequential patterns were captured for each malware sample.
  - **Example dataframe showing frequency of n-grams:**

| | XADD.LOCK | XCHG | XGETBV | XLAT | XOR | XORPD | XORPS | Y |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 98 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 21 | 0 | 0 | 0 |
| 2 | 0 | 3 | 0 | 0 | 1387 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**7. Train-Test Split**

- APT group labels, represented as directory names, were converted to numerical labels using a label encoder. This step was necessary for compatibility with machine learning classifiers.
- The dataset was split into training and testing subsets, with 80% of the data used for training the classifier and 20% reserved for evaluation. This split ensured the ability to assess the classifier's generalization performance on unseen data.

**Hybrid Oversampling with ROS and SMOTE**

The dataset initially exhibited class imbalances, with some APT groups being heavily underrepresented compared to others. Class imbalance can lead to a classifier favoring the majority class, reducing accuracy on minority classes and potentially overlooking less common but significant malware types (Kamalov et al., 2023). To address this, Random Oversampling

was applied, which duplicates samples from minority classes until each class is represented equally in the training data.

The benefits of random oversampling include increasing the model's exposure to minority class samples and simplifying the model training as each class is equally represented (Kamalov et al., 2023).
The negative tradeoff of random oversampling is the risk of overfitting, where duplicate samples may result in the classifier "memorizing" minority class examples. There is also an increase in training time and resource demands due to the expanded dataset size.

To address the severe class imbalance in the dataset, a two-step hybrid approach was employed:

1. **Random OverSampling (ROS)**: The RandomOverSampler was applied to handle extreme cases where classes had very few samples, ensuring that all classes had a minimum number of representatives. ROS simply duplicates minority class samples to increase their presence in the dataset (Kamalov et al., 2023).
    - **Why ROS First?**
        - It mitigates the challenge of SMOTE failing due to classes with a single sample. SMOTE relies on creating synthetic samples by interpolating between nearest neighbors, which is not feasible if only sample in a class.
2. **Synthetic Minority Oversampling Technique (SMOTE)**: After ROS, SMOTE was applied to generate synthetic samples for minority classes by interpolating between existing samples within the same class. This approach creates more diverse examples while avoiding exact duplication of minority class samples (Alkhawaldeh et al., 2023).
    - **SMOTE Parameters**:
        - `random_state=42`: Used for reproducibility.
        - `k_neighbors=5`: Specified that SMOTE uses the 5 nearest neighbors to generate synthetic samples.

## Results

The following classifiers were tested: Support Vector Machine (SVM), K-Nearest Neighbors (KNN) (tested with k=3 and k=5 to evaluate accuracy with varying neighborhood sizes), and Decision Tree. Each classifier was trained on the balanced datasets, improving their ability to generalize across all malware types.

The classifiers were evaluated using accuracy as the primary metric. The following table outlines the accuracy, precision, recall, and F1-scores of the classifiers trained on the pre-balanced data using the combined features (1-gram and 2-gram), contrasted against the classifiers trained on the post-balanced data set with combined features.

| Classifier | Pre-Balance Accuracy | Pre-Balance Precision | Pre-Balance Recall | Pre-Balance F1-Score | Post-Balance Accuracy | Post-Balance Precision | Post-Balance Recall | Post-Balance F1-Score |
|---|---|---|---|---|---|---|---|---|
| **SVM** | 0.3023 | 0.0400 | 0.0800 | 0.0500 | **0.7361** | **0.8595** | **0.7361** | **0.7188** |
| **KNN-3** | 0.4651 | 0.200 | 0.2600 | 0.2100 | **0.8958** | **0.8723** | **0.8958** | **0.8690** |
| **KNN-5** | 0.2791 | 0.1200 | 0.1500 | 0.1200 | **0.8958** | **0.8746** | **0.8958** | **0.8695** |
| **Decision Tree** | 0.3953 | 0.1400 | 0.1800 | 0.1600 | **0.5000** | **0.4142** | **0.5000** | **0.4307** |

**Figure 1: Classifier Results for Combined Data (1-gram + 2-gram)**

| Classifier | 1-Gram Accuracy | 1-Gram Precision | 1-Gram Recall | 1-Gram F1-Score | 2-Gram Accuracy | 2-Gram Precision | 2-Gram Recall | 2-Gram F1-Score |
|---|---|---|---|---|---|---|---|---|
| **SVM** | 0.1163 | 0.1100 | 0.0800 | 0.0900 | 0.0698 | 0.0900 | 0.0800 | 0.0700 |
| **KNN-3** | 0.4186 | 0.2200 | 0.2200 | 0.2100 | 0.3256 | 0.1100 | 0.1300 | 0.1000 |
| **KNN-5** | 0.3721 | 0.1900 | 0.2000 | 0.1900 | 0.2791 | 0.1700 | 0.1700 | 0.1500 |
| **Decision Tree** | 0.4419 | 0.2500 | 0.2400 | 0.2300 | 0.3488 | 0.1300 | 0.1400 | 0.1300 |

**Figure 2: Classifier Results for 1-gram vs. 2-gram Opcode Training Sets**

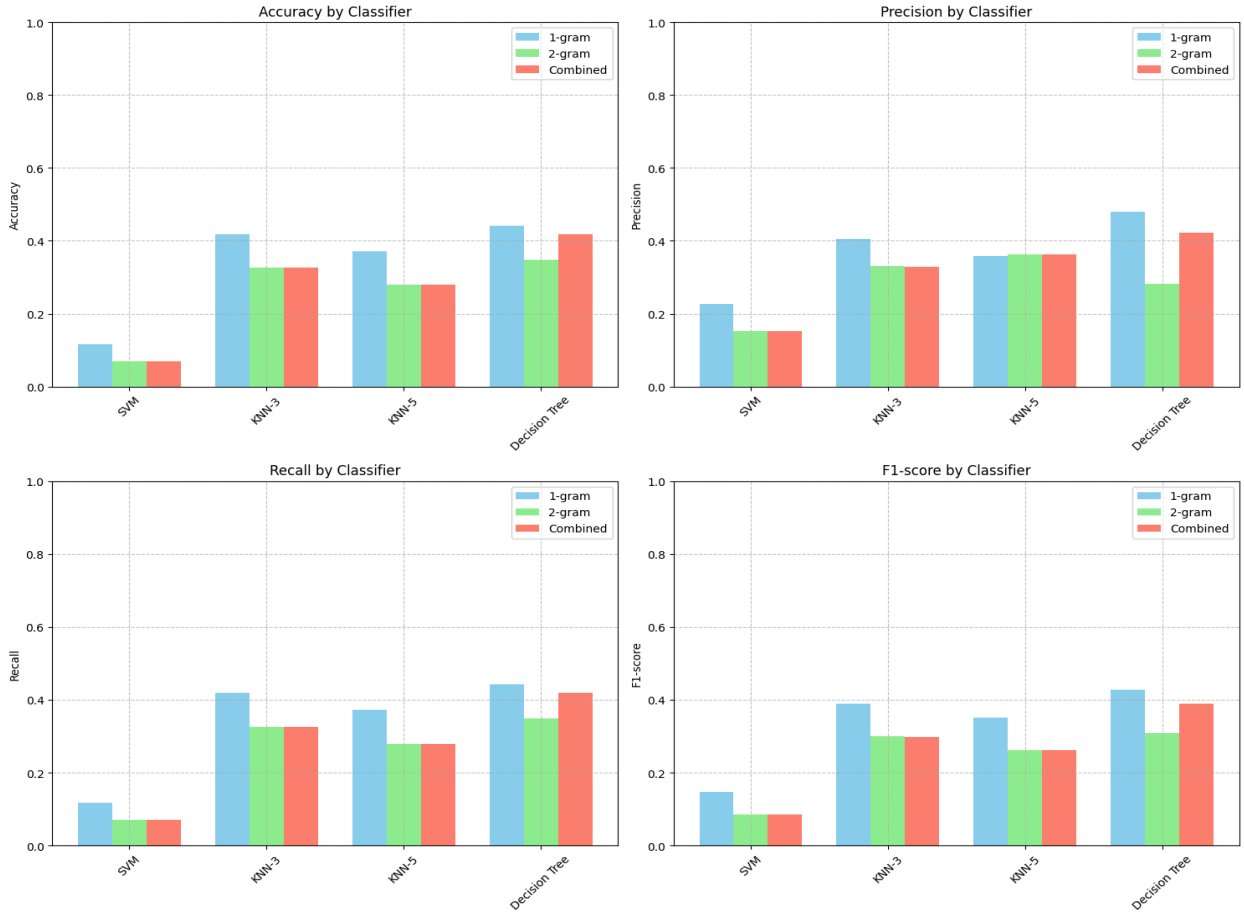


**Figure 3: Results for Classifiers Trained on Unbalanced Data**

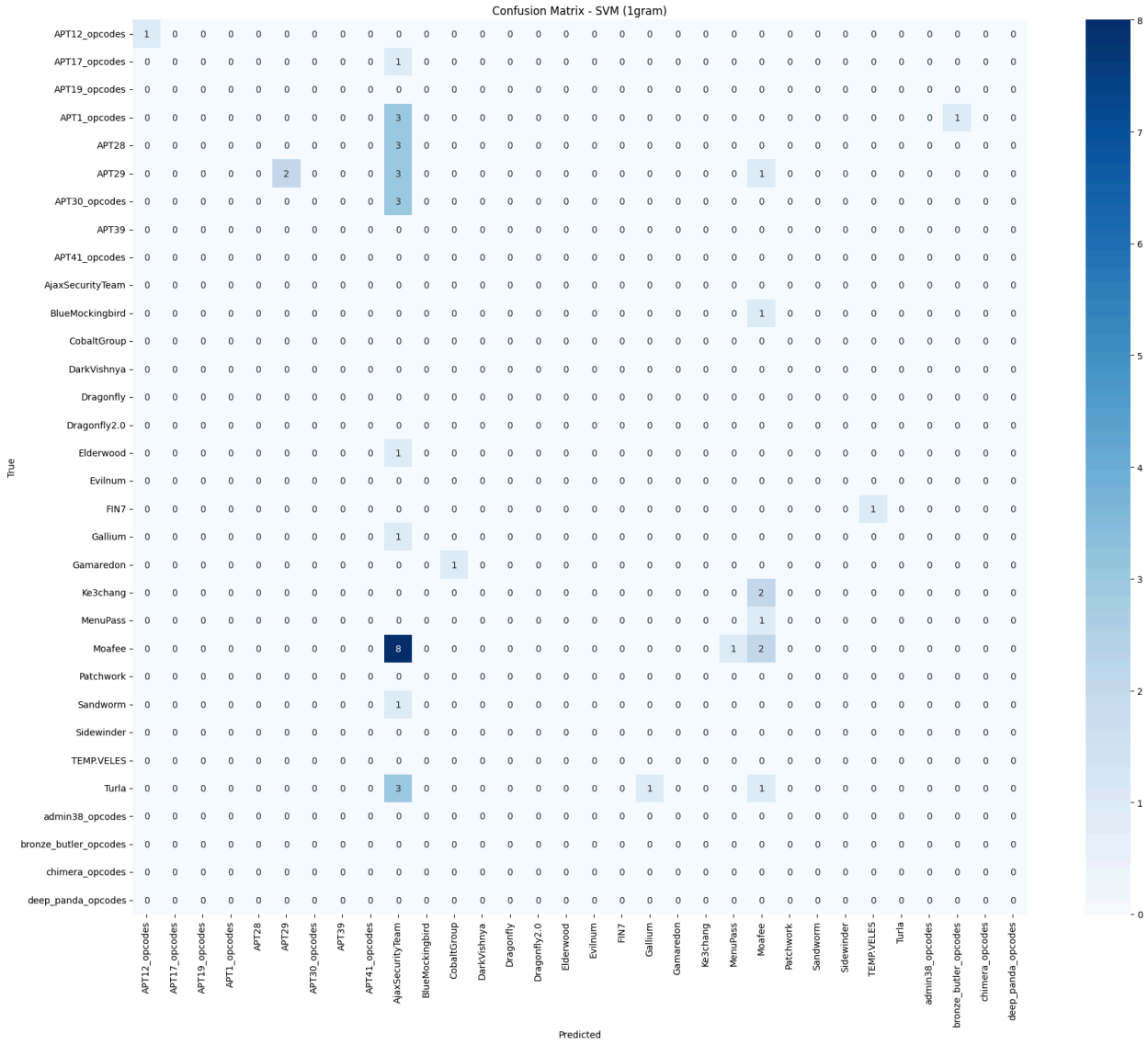


**Figure 4: Confusion matrix for SVN trained on 1-gram opcodes from unbalanced data set**

**Pre-Balancing Results:**

Before applying class balancing, the classifiers showed reduced accuracy, particularly in identifying malware types from underrepresented classes (see Figure 1). This trend was especially evident in the Decision Tree and SVM classifiers, which performed suboptimally on the minority classes. This suggests that without balancing, the classifiers lacked the diversity in training data necessary for accurate prediction across all classes. The results of 1-gram and 2-gram features are shown in **Figure 2.** The SVM confusion matrix for the 1-gram (**Figure 4)** results further highlight the propensity of the models to mislabel the opcode samples due to the limited data. To see all of the confusion matrices for each classifier across 1-gram, 2-gram, and combined feature sets, please see the Jupyter Notebook (apt_classification.ipynb). The bar chart in **Figure 3** summarizes the metrics for these results.

To improve the model and address the issue of having an unbalanced number of malware samples across the APT groups, the aforementioned ROS + SMOTE technique was used, as well as combining the 1-gram and 2-gram feature sets when training the models. This technique significantly improved the accuracy of the SVN and KNN classifiers, as seen in **Figure 6.**

**Post-balancing Results**

- **K-Nearest Neighbors (k=3 and k=5):** After balancing the training data with ROS + SMOTE, this classifier achieved the highest accuracy across 1-gram, 2-gram, and combined features, achieving an accuracy of TODO
- **Decision Tree:** This classifier had the lowest accuracy, and did not benefit from the balanced data set as much as the other classifiers. This highlights the Decision Tree algorithm's sensitivity to data imbalance and propensity to overfit when exposed to oversampled data.
- **SVM:** This classifier maintained moderate accuracy across all feature types, comparable to but not surpassing KNN in this dataset.

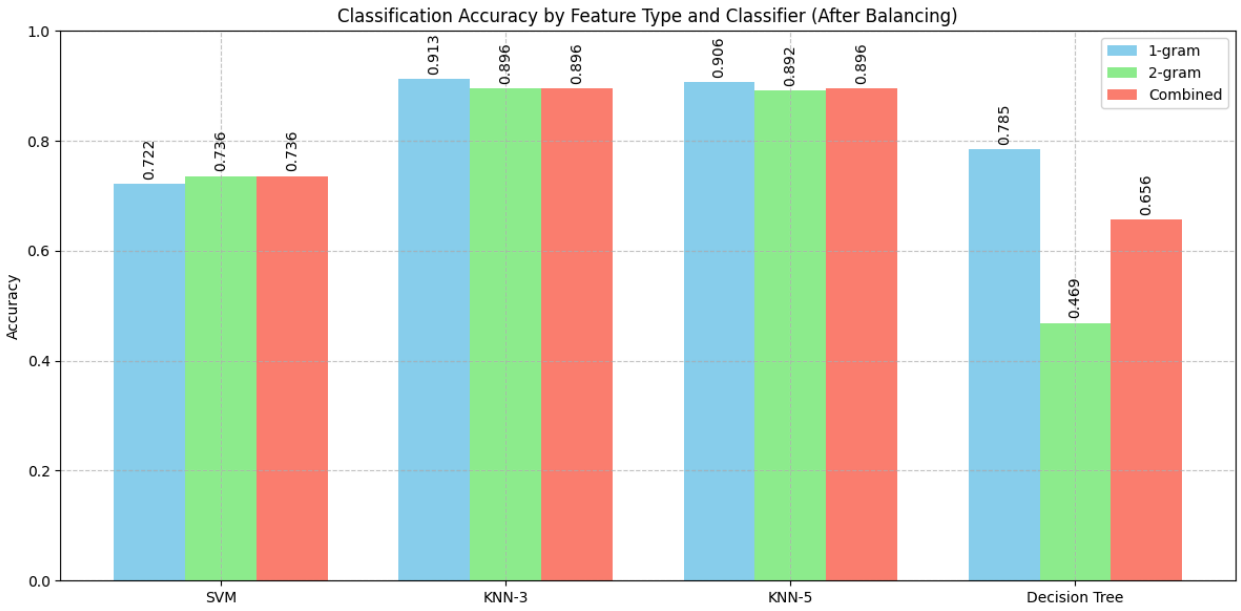


**Figure 6: Accuracy of Classifiers Trained on Balanced Data**

## Conclusion

The results indicate that K-Nearest Neighbors with k=3 performs best, achieving over 91% accuracy on 1-gram features. Balancing the classes through random oversampling improved classifier performance on minority classes.

**References:**

Bragen, S. R. (2015). Malware detection through opcode sequence analysis using machine learning. https://www.semanticscholar.org/paper/Malware-detection-through-opcode-sequence-analysis-Bragen/62f796c19ffa2ee70fc5ee7aec0fe41fae26f191

Alkhawaldeh, I. M., Albalkhi, I., & Naswhan, A. J. (2023). Challenges and limitations of synthetic minority oversampling techniques in machine learning. World Journal of Methodology, 13(5), 373–378. https://doi.org/10.5662/wjm.v13.i5.373

Kamalov, F., Leung, H., & Cherukuri, A. K. (2023). Keep it simple: random oversampling for imbalanced data. 2023 Advances in Science and Engineering Technology International Conferences, ASET 2023, 1–4. https://doi.org/10.1109/aset56582.2023.10180891