Aprendizagem Aplicada à Segurança

Malware Detection as a Multi-Class Problem

Mário Antunes

November 14, 2025

Universidade de Aveiro

Table of Contents i

The Problem: Beyond Binary

- **Binary Classification:** Is this file `malicious` or `benign`? (A "yes/no" question).
- **Multi-Class Classification:** *What kind* of malware is this? (A "which one?" question).

This is a **multi-class** problem. We're not just detecting *if* a file is bad, but trying to assign it to one of K distinct families.

Example Classes: 1. `Benign` 2. `Trojan` 3. `Worm` 4. `Ransomware` 5. `Spyware` 6. `Adware` 7. (...and so on)

This requires different modeling and evaluation techniques than binary classification.

The Pipeline: From File to Family

1. **Data Ingestion:** Start with a collection of raw files (e.g., `.exe`, `.dll`) and their known labels (malware family).
2. **Feature Engineering:** Convert these raw binary files into a numerical vector format. This is the most critical step.
3. **Model Training:** Train a multi-class classifier to learn the patterns that map a file's "features" to its "family."
4. **Model Evaluation:** Use specialized multi-class metrics to see how well the model performs and *where* it makes mistakes.

The Core Challenge: Feature Engineering

How do you turn a 2MB binary file into a set of numbers (a vector)? We can't use the raw bytes. We must extract **features**.

Common Approaches:

- **Static Analysis (Used Here):** Analyzing the file *without running it*.
    - **Byte-Level N-grams:** The primary technique.
    - **PE File Headers:** Metadata about the file (compilation time, imported libraries, etc.).
    - **String Analysis:** Extracting human-readable strings from the binary.
- **Dynamic Analysis (Not Used Here):** Running the file in a safe "sandbox" to see what it *does* (e.g., "tries to delete files," "contacts a server").

Feature Engineering: Byte-Level N-grams

This is the "Bag of Words" equivalent for binary files.

- **N-gram:** A sequence of N items.
  - **Text 2-gram:** `"hello"` -> `(he, el, ll, lo)`
  - **Byte 2-gram:** A file's byte sequence `A3 4F C1` -> `(A3 4F, 4F C1)`

The Process: 1. Read the entire file as a sequence of bytes. 2. Slide a "window" of size N across the sequence, creating millions of N-grams. 3. We treat each unique N-gram (e.g., `4F C1`) as a "word" in our vocabulary. 4. This creates a **massive vocabulary** (millions of features).

Why N-grams? They capture small, recurring "byte-patterns" that are characteristic of specific malware families (e.g., a specific decryption loop or exploit code).

Feature Engineering: Handling Massive Features i

A vocabulary of 5 million N-grams is too large. We need to reduce this.

1. **TF-IDF (Term Frequency-Inverse Document Frequency)**
   - This is a weighting scheme, not just a count.
   - $TF - IDF(t, d) = TF(t, d) \times IDF(t)$
   - **Term Frequency (TF):** "How many times does N-gram `t` appear in file d?"
   - **Inverse Document Frequency (IDF):** "How rare is N-gram `t` across *all* files?"

Feature Engineering: Handling Massive Features ii

- **Intuition:** This finds N-grams that are **frequent in one file** (or one family) but **rare everywhere else**. It boosts the signal of unique, malicious byte patterns and filters out common, benign patterns (like standard library code).

2. **Feature Hashing**
   - A fast, memory-efficient way to map a huge feature space to a smaller, fixed-size vector.
   - It uses a hash function to map N-grams to column indices.
   - **Pro:** Very fast, no vocabulary storage needed.

Feature Engineering: Handling Massive Features iii

- **Con:** Can have "hash collisions" (different N-grams map to the same index), which can add noise.

The Multi-Class Output: Softmax i

In binary classification, a sigmoid function outputs *one* probability ($P(y = 1)$).

In multi-class, we need *K* probabilities (one for each class), and they must all sum to 1. This is done by the **Softmax** function.

Softmax Function: * Takes a vector of raw scores (logits) from the model. * Converts them into a probability distribution. * $P(class_i) = \frac{e^{score_i}}{\sum_{j=1}^{K} e^{score_j}}$

The Multi-Class Output: Softmax ii

Example: * **Model Scores:** `[Trojan: 2.7, Worm: -1.0, Benign: 0.5]` * **Softmax Probs:** `[Trojan: 0.88, Worm: 0.02, Benign: 0.10]` * The model's prediction is **Trojan**.

This is the default output layer for all multi-class neural networks and is used by Logistic Regression in its multi-class (non-OvR) form.

The Multi-Class Loss Function i

How do we measure the error of a Softmax output?

We use **Categorical Cross-Entropy** (or "Log Loss").

The Formula (for one example):

- $Loss = -\sum_{i=1}^{K} y_i \log(p_i)$
- y_i is the "true" label (a one-hot vector, e.g., `[1, 0, 0]`).
- p_i is the model's predicted probability (from Softmax, e.g., `[0.88, 0.02, 0.10]`).

Intuition:

- The loss is simply $-\log(p_{true\_class})$.

The Multi-Class Loss Function ii

- It only looks at the probability the model assigned to the **correct class**.
- If $p_{true} = 0.88$ (confident & correct) -> $Loss = -\log(0.88) = 0.12$ (low loss).
- If $p_{true} = 0.02$ (unconfident & wrong) -> $Loss = -\log(0.02) = 3.91$ (high loss).
- This loss function forces the model to put all its "probability mass" on the correct class.

Modeling: Classic Approaches i

1. **Logistic Regression**
   - A linear model, but powerful and a great baseline.
   - By default, it uses a **One-vs-Rest (OvR)** strategy for multi-class.
     - It trains K separate binary classifiers (e.g., `Trojan vs. Not-Trojan`, `Worm vs. Not-Worm`, ...).
     - The class with the highest confidence score wins.
   - Can also be configured to use a **Softmax** output directly.

Modeling: Classic Approaches ii

2. **Naive Bayes (Multinomial)**
   - A probabilistic model based on Bayes' Theorem.
   - It's "naive" because it assumes all features (N-grams) are independent.
   - **Multinomial Naive Bayes (MNB)** is perfect for this task because it's designed to work with **counts** (like N-gram counts or TF-IDF scores).
   - It's extremely fast and often hard to beat for text/byte classification.

3. **Random Forest / Gradient Boosting (XGBoost)**
   - **Ensemble** methods that combine many “weak” decision trees into a single “strong” model.
   - **Random Forest:** Builds many trees in parallel on random subsets of data/features. Averages their votes.
   - **XGBoost:** Builds trees *sequentially*, where each new tree corrects the errors of the previous ones.
   - **Pros:** Very powerful, non-linear (can find complex patterns), provides “feature importance” (we can see which N-grams are most predictive).

4. **Multi-Layer Perceptron (MLP)**
   - A simple neural network.
   - **Structure:**
     - **Input Layer:** The TF-IDF vector.
     - **Hidden Layers:** One or more layers with `ReLU` activation to learn complex, non-linear combinations of features.
     - **Output Layer:** A `Softmax` layer with K neurons to output the class probabilities.
   - **Training:** Uses the **Categorical Cross-Entropy** loss function.

Evaluation: The K x K Confusion Matrix i

This is the **most important** tool for multi-class evaluation. * It's an $N \times N$ matrix where N is the number of classes. * The **rows** are the **Actual** (True) classes. * The **columns** are the **Predicted** classes.

	Pred: Trojan	**Pred: Worm**	**Pred: Benign**
Actual: Trojan	**250 (TP)**	5 (FN)	2 (FN)
Actual: Worm	8 (FN)	**150 (TP)**	0 (FN)
Actual: Benign	3 (FP)	1 (FP)	**500 (TP)**

Evaluation: The K x K Confusion Matrix ii

What it tells us:

- The **diagonal** (bold) is what we got **right** (True Positives for each class).
- **Off-diagonal** numbers are **errors**.
- **Look for patterns!** "The model correctly identifies `Benign` files (500/504), but it often confuses `Trojan` (8) for `Worm`." This tells you *where* your model is failing.

Evaluation: Multi-Class Metrics (Averaging) i

How do you calculate a single F1-Score for K classes? You have to average.

Accuracy: * $\frac{\text{Sum of Diagonal}}{\text{Total Samples}}$ * Easy to understand, but **very misleading** if the classes are **imbalanced** (e.g., 10,000 Benign files vs. 50 Trojan files).

Precision, Recall, F1-Score: We must choose an *averaging strategy*.

- **Micro Average:**
    - **How:** Sum all TPs, FPs, and FNs *globally* across all classes, *then* calculate the metric.

Evaluation: Multi-Class Metrics (Averaging) ii

- **What it means:** It's a "globally" correct metric. It is heavily biased by the **most common class**. It's effectively a weighted-by-size metric.

- **Macro Average:**
    - **How:** Calculate the metric (e.g., F1-Score) for *each class independently*, then take the simple average of those scores.
    - **What it means:** It gives **equal weight to every class**, no matter how rare. This is the *best* metric for imbalanced datasets if you care about performance on rare classes.
- **Weighted Average:**
    - **How:** Same as Macro, but the final average is weighted by the number of samples in each class.

Evaluation: Multi-Class Metrics (Averaging) iii

- **What it means:** A good compromise. It's often very close to the Micro-average.

For this problem, Macro-F1 is the most honest measure of performance.