

Malware Detection and Classification Using PCA, k-Nearest Neighbors, and Decision Trees

CYSE 420 — Applied Machine Learning in Cybersecurity
Assignment 03 | Carl Lochstampfor Jr. | Fall 2026

1. Introduction and Dataset

This report documents the construction and comparison of two supervised classifiers for static malware detection. Both models operate on Portable Executable (PE) header fields extracted from Windows binaries, classifying each file as malicious or benign without executing it. The pipeline consists of data cleaning, feature selection, standardization, dimensionality reduction via Principal Component Analysis, and evaluation of a k-Nearest Neighbors classifier against a Decision Tree classifier.

The dataset, **file_pe_headers.csv**, contains 19,611 samples described by 79 columns: a file identifier, 77 numeric PE header features, and a binary **Malware** label. Feature fields span the DOS header (e_magic, e_lfanew), the COFF file header (Machine, NumberOfSections, TimeDateStamp), the optional header (SizeOfCode, ImageBase, Subsystem, CheckSum), and derived section statistics (SectionMinEntropy, SectionsLength, SuspiciousImportFunctions). No values are missing. As delivered, the class distribution is 14,599 malicious to 5,012 benign, or 74.4 percent malicious.

2. Data Preprocessing and Feature Selection

2.1 Removal of an identifier-based label leak

The **Name** column was removed before any modeling. Inspection showed that benign samples carry genuine Windows filenames (for example, ACCTRES.dll) while every malicious sample carries a VirusShare_ prefix followed by a hash. A cross-tabulation confirmed the separation is perfect: all 5,012 benign records and none of the 14,599 malicious records lack the prefix. Retaining the column would allow a trivial string rule to achieve 100 percent accuracy while learning nothing about executable structure, and such a rule would not generalize to files encountered outside a malware repository.

2.2 Duplicate records and what they reveal

With **Name** present, no duplicate rows existed, because each identifier is unique. After removing it, 2,980 duplicate records appeared. Of these, 2,934 (98.5 percent) were malicious and only 46 were benign.

This asymmetry is informative rather than incidental. Each duplicate pair represents files with different hashes but byte-identical PE headers. That is the expected signature of polymorphism and packing: repacking a sample alters its body and therefore its hash, while the header structure emitted by the same builder or packer remains constant. Retaining these records would allow the same header signature to appear in both the training and test partitions, inflating measured performance through memorization rather than generalization. After removal, the working dataset contains 16,631 samples and the class balance shifts to 70.1 percent malicious,

which becomes the majority-class baseline that any model must exceed.

2.3 Partitioning

The data was split 70/30 into 11,641 training and 4,990 test samples, stratified on the target so both partitions preserve the 70.1/29.9 ratio, with a fixed random seed for reproducibility. The split precedes all subsequent transformations. Variance thresholding, standardization, and PCA each estimate parameters from data; fitting any of them on the full dataset would transfer information from the test partition into the training pipeline. Each was therefore fitted on the training set alone and applied to both partitions.

2.4 Variance threshold feature selection

Feature selection used a variance threshold of zero, removing only features whose value never changes. Eight of 77 features were removed, leaving 69: `e_magic`, `SectionMaxEntropy`, `SectionMaxRawsize`, `SectionMaxVirtualsize`, `SectionMinPhysical`, `SectionMinVirtual`, `SectionMinPointerData`, and `SectionMainChar`.

A zero threshold rather than a tuned value is a deliberate choice. Variance is scale-dependent: `ImageBase` is measured in units of 10^{11} while `Subsystem` takes single-digit values, so any positive threshold applied to unscaled data would eliminate small-magnitude features irrespective of their predictive value. Standardizing first does not resolve this, since standardization sets every feature variance to exactly one, rendering the filter inert. Zero is therefore the only threshold defensible without prior scaling. This limitation is intrinsic to the method, which evaluates features without reference to the target variable.

Three of the removed features indicate a data collection fault rather than a natural constant. `SectionMaxEntropy`, `SectionMaxRawsize`, and `SectionMaxVirtualsize` are zero throughout while their corresponding minimum-valued counterparts contain varied measurements. A maximum cannot fall below its own minimum, so these fields were not populated correctly when the dataset was assembled. By contrast, `e_magic` is a legitimate constant: the value 23117 encodes the ASCII signature "MZ" that begins every valid PE file by specification.

2.5 Standardization

Each remaining feature was converted to a z-score using training-set means and standard deviations. The necessity is evident from the raw magnitudes: `ImageBase` has a standard deviation of 5.21×10^{11} against 0.51 for `Subsystem`, a difference of roughly twelve orders of magnitude. kNN classifies by Euclidean distance, which on unscaled data would be determined almost entirely by `ImageBase`. PCA maximizes captured variance and would return components tracking only the largest-magnitude columns. After standardization every feature contributes on comparable terms.

3. Dimensionality Reduction with PCA

Principal Component Analysis was fitted on the 69 standardized training features to construct orthogonal components ordered by captured variance. The variance profile is notably flat. The first component captures 11.0 percent and the second 6.5 percent, for a combined 17.6 percent; the first ten together reach only 47.5 percent. Retaining 80 percent of total variance requires 30 components, 90 percent requires 40, and 95 percent requires 47.

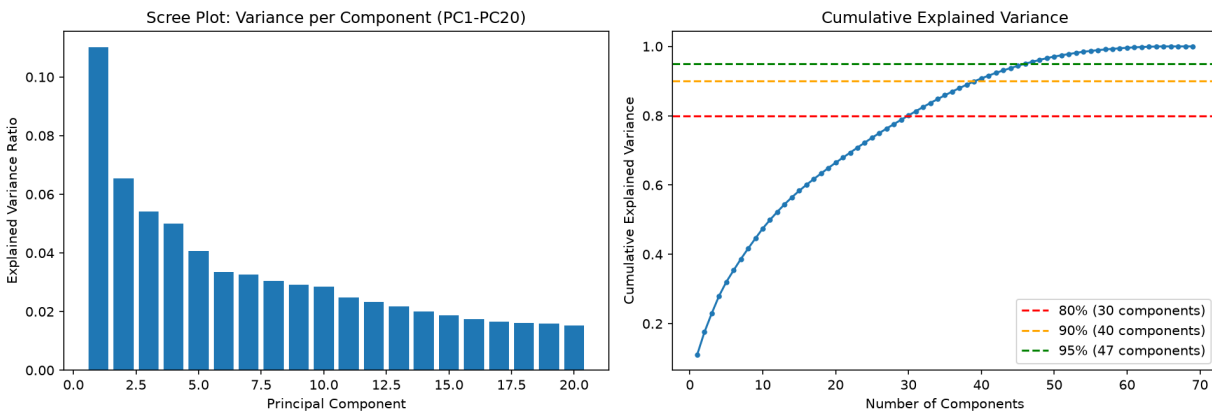


Figure 1. Scree plot of individual explained variance (left) and cumulative explained variance with 80, 90, and 95 percent reference lines (right).

Typical PCA demonstrations show the first two or three components capturing 60 to 70 percent of variance. The flatter profile here indicates that PE header fields measure largely independent file properties: compilation timestamps, section entropy, subsystem type, and size fields do not co-vary strongly, so no small set of dominant directions exists. The scree plot shows a correspondingly soft elbow near the fifth component rather than a sharp one, and the cumulative curve rises steadily instead of levelling off early.

Thirty components were retained, preserving 80.07 percent of variance while reducing dimensionality by 57 percent. This representation was used to train both classifiers.

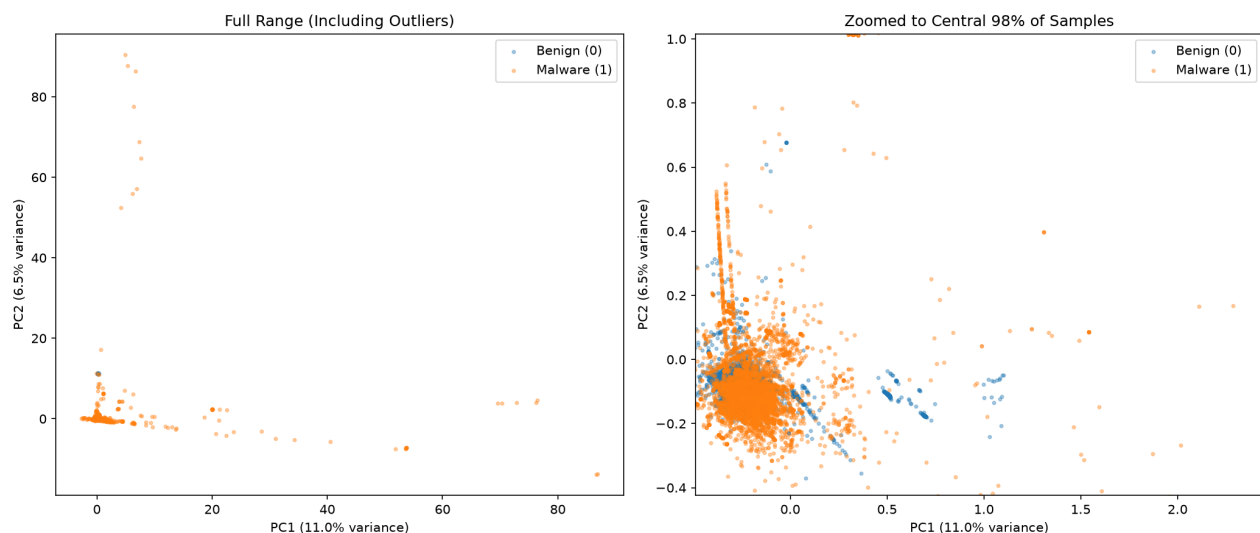


Figure 2. Training data projected onto the first two principal components, shown at full range (left) and restricted to the central 98 percent of samples on each axis (right).

A separate two-component projection was fitted for visualization. The full-range view is dominated by a small number of extreme samples that stretch both axes beyond 80, compressing 96.9 percent of the data into an indistinguishable mass at the origin; the second panel restricts the display range without removing any data.

The zoomed view shows the classes overlapping substantially but not uniformly. Benign samples concentrate near the origin and several form tight isolated clusters along the first component, consistent with legitimate software produced by a small number of standard compilers and build toolchains. Malicious samples occupy that same central region while extending far beyond it in every direction, accounting for all extreme outliers. Malware thus occupies a substantially larger volume of the feature space than benign software, which is the expected consequence of packing and obfuscation producing non-standard header values. No linear boundary separates the classes in this projection. This figure retains only 17.6 percent of total variance, so it necessarily understates the separability available to models operating on 30 components.

4. k-Nearest Neighbors Implementation

kNN classifies a sample by locating its k nearest training points in Euclidean distance and taking a majority vote among their labels. It performs no training-time optimization, storing the training set and deferring all computation to prediction time.

Eight odd values of k were evaluated, scored by F1 rather than accuracy given the class imbalance. Odd values avoid tied votes in binary classification.

k	1	3	5	7	9	11	15	21
F1	0.9808	0.9823	0.9813	0.9810	0.9802	0.9790	0.9775	0.9763

Table 1. Test-set F1 across candidate values of k .

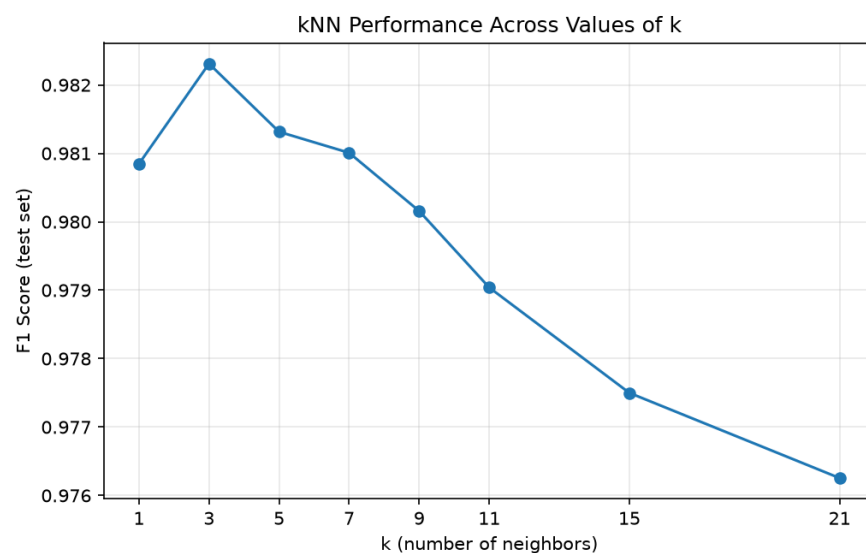


Figure 3. kNN F1 score as a function of k . Note the narrow y-axis range.

F1 peaks at $k = 3$ and declines monotonically thereafter. The total spread across all eight candidates is 0.006, so the choice has limited practical consequence, and the narrow vertical axis in Figure 3 visually exaggerates the decline. The direction of the trend nonetheless carries information: performance falling as k increases indicates the class signal is highly local, such that a sample's three nearest neighbors predict its label well while widening the vote to 21 begins incorporating samples from outside its immediate group. This is consistent with the tight clustering visible in Figure 2. That $k = 1$ scores 0.9808, nearly matching the optimum despite being the most overfit possible configuration, reinforces the same conclusion. $k = 3$ was selected.

5. Decision Tree Implementation

A decision tree classifies by learning a sequence of threshold tests on individual features, selecting each split by the reduction in Gini impurity it produces. The mechanism differs fundamentally from kNN: the tree performs its work during training and predicts by traversing a short path of comparisons. It is also scale-invariant, since a threshold test is unaffected by rescaling an axis, though the same standardized PCA representation was used here so that both models receive identical inputs.

An unconstrained tree reached depth 23 with 304 leaves and a training F1 of 0.9998, indicating near-complete memorization. Constraining maximum depth produces the characteristic overfitting curve.

Max depth	3	5	7	10	15	None (23)
Leaves	8	30	77	140	225	304
Train F1	0.9609	0.9711	0.9834	0.9900	0.9962	0.9998
Test F1	0.9552	0.9658	0.9758	0.9761	0.9752	0.9737
Gap	0.0057	0.0053	0.0076	0.0139	0.0210	0.0261

Table 2. Decision tree performance against depth constraint.

Training F1 rises monotonically with depth while test F1 peaks between depths 7 and 10 and then declines. The gap between them widens from 0.0076 at depth 7 to 0.0261 unconstrained, so the deepest tree generalizes worse than one less than a third its depth. Depth 7 was selected: it matches the best observed test F1 to within 0.0003 while using 77 leaves rather than 140 and carrying approximately half the train/test gap. Where two configurations perform equivalently on held-out data, the smaller model is preferable, being less dependent on training-set specifics.

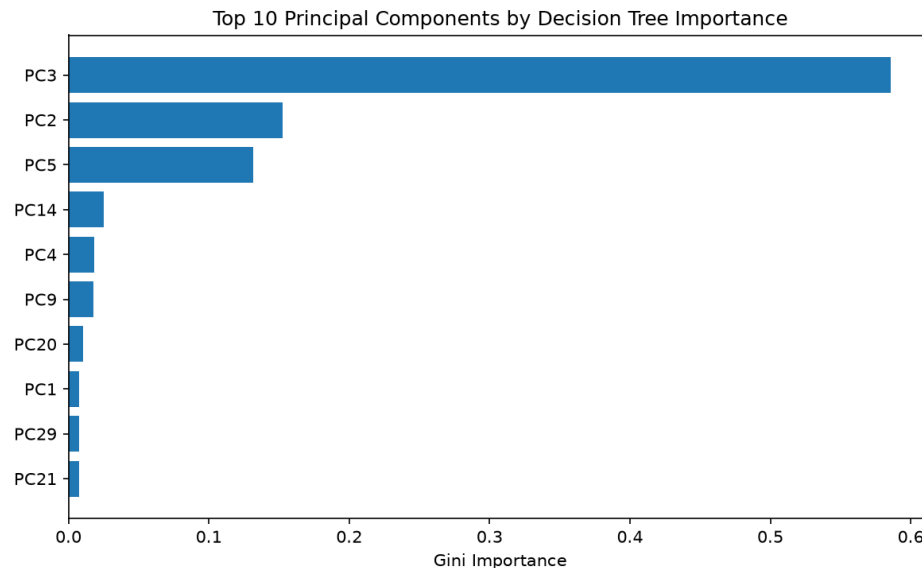


Figure 4. Gini importance of the ten most influential principal components.

The tree's use of the available components is highly concentrated. The third component alone accounts for roughly 59 percent of total importance, with the second (15 percent) and fifth (13 percent) bringing the top three to approximately 87 percent. The top ten reach 96.4 percent, and six of the thirty components are never used.

The most instructive result is that the first principal component ranks eighth in importance despite capturing the largest share of variance in the data. This exposes a distinction between variance and class relevance. PCA is unsupervised: it orders components by how much samples vary along each direction, without reference to the labels. A high-variance direction need not be discriminative. The first component most likely tracks a

property along which all files differ substantially regardless of class, such as overall file size, while the third captures less total variance but aligns more closely with the malicious/benign distinction. This is the same limitation identified for variance-threshold feature selection in Section 2.4, arising in a different technique for the same underlying reason.

6. Model Evaluation and Comparison

Both models were evaluated on the same held-out set of 4,990 samples, comprising 3,500 malicious and 1,490 benign files.

Model	Accuracy	Precision	Recall	F1	ROC AUC
kNN (k = 3)	0.9752	0.9809	0.9837	0.9823	0.9840
Decision Tree (depth 7)	0.9661	0.9776	0.9740	0.9758	0.9639

Table 3. Test-set performance. Majority-class baseline accuracy is 0.7014.

kNN outperforms the decision tree on every metric. Accuracy is the least informative of them: a classifier predicting "malicious" for all samples would score 0.7014 while identifying no benign software whatsoever, and both models exceed that baseline by margins accuracy cannot usefully distinguish.

Recall and precision carry distinct operational costs in this application. Recall measures the proportion of malicious files detected, so a false negative is a threat delivered to the endpoint. Precision measures how many flagged files were genuinely malicious, so a false positive is legitimate software blocked or quarantined. The appropriate balance depends on deployment context: an endpoint scanner presenting alerts to an analyst may accept false positives in order to maximize recall, whereas an automated quarantine system removing files without review must weigh precision more heavily.

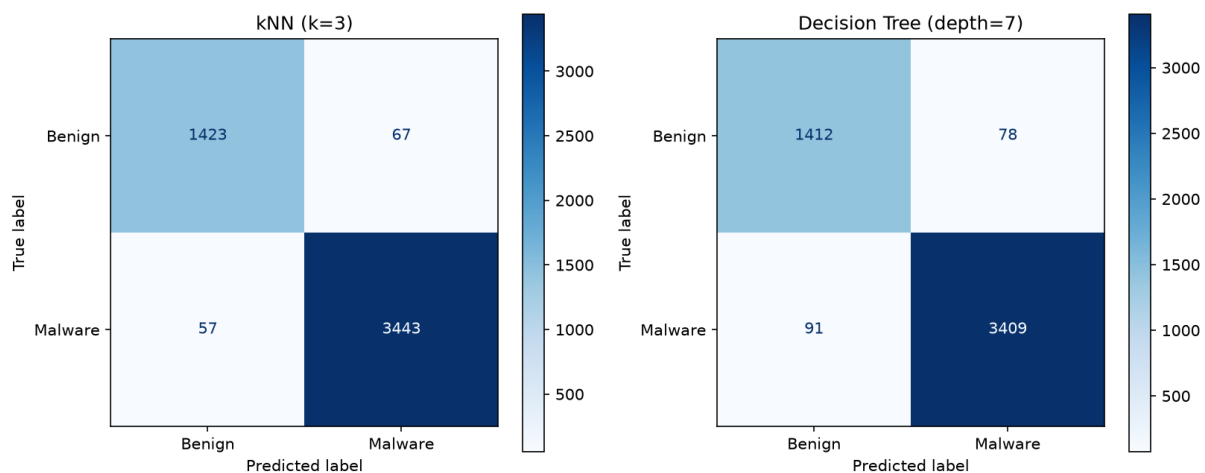


Figure 5. Confusion matrices. True labels on rows, predictions on columns.

Expressed in absolute counts, kNN missed 57 malicious files and misclassified 67 benign files. The decision tree missed 91 and misclassified 78. The tree's disadvantage is concentrated in false negatives, missing 60 percent more malicious files while its false positive count rose by only 16 percent. Since a false negative represents an undetected threat, this is the more consequential error type for a detection system, and it is the error the tree makes disproportionately more often.

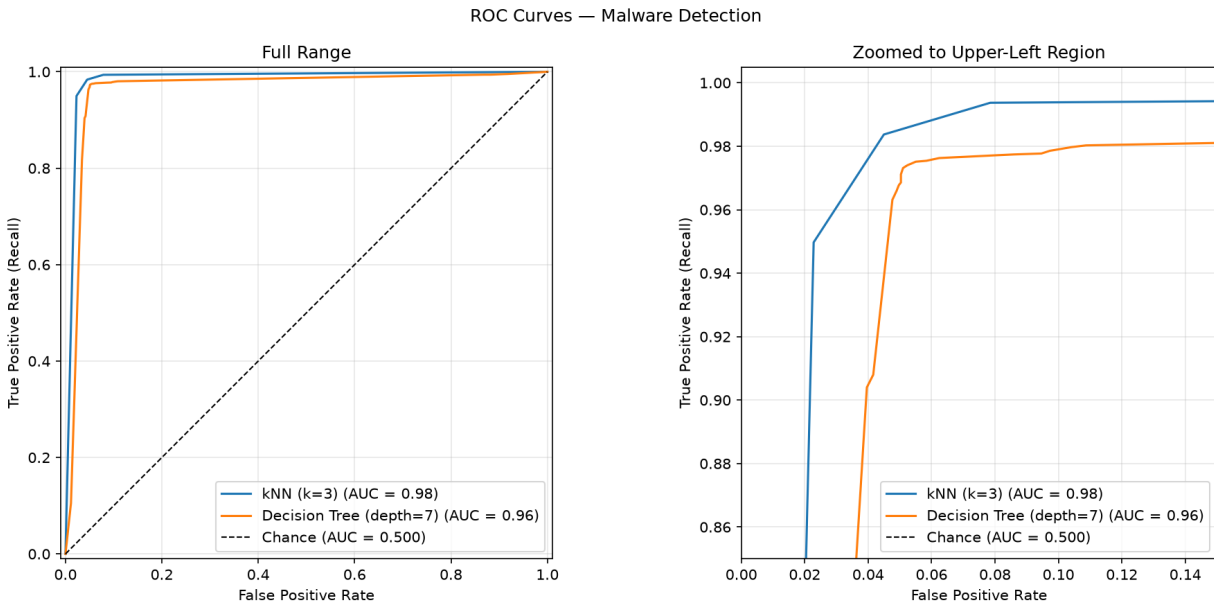


Figure 6. ROC curves at full range (left) and restricted to the upper-left region (right).

The ROC AUC gap of 0.020 is proportionally wider than the F1 gap of 0.0065. AUC evaluates ranking quality across all decision thresholds rather than performance at the default 0.5 cutoff, indicating the tree's ranking is weaker than its hard predictions alone suggest.

This holds despite the tree possessing finer probability resolution. With $k = 3$, kNN can emit only four distinct probability values (0, $1/3$, $2/3$, and 1) and therefore offers only a handful of usable operating points, visible as long straight segments in the zoomed panel. The tree draws on 25 distinct leaf probabilities across the test set. That the higher-resolution model produces the lower curve indicates its leaf-level confidence is poorly calibrated, assigning high certainty to a subset of samples it classifies incorrectly; the additional thresholds do not recover performance.

6.1 Structural Analysis: Decision Boundaries

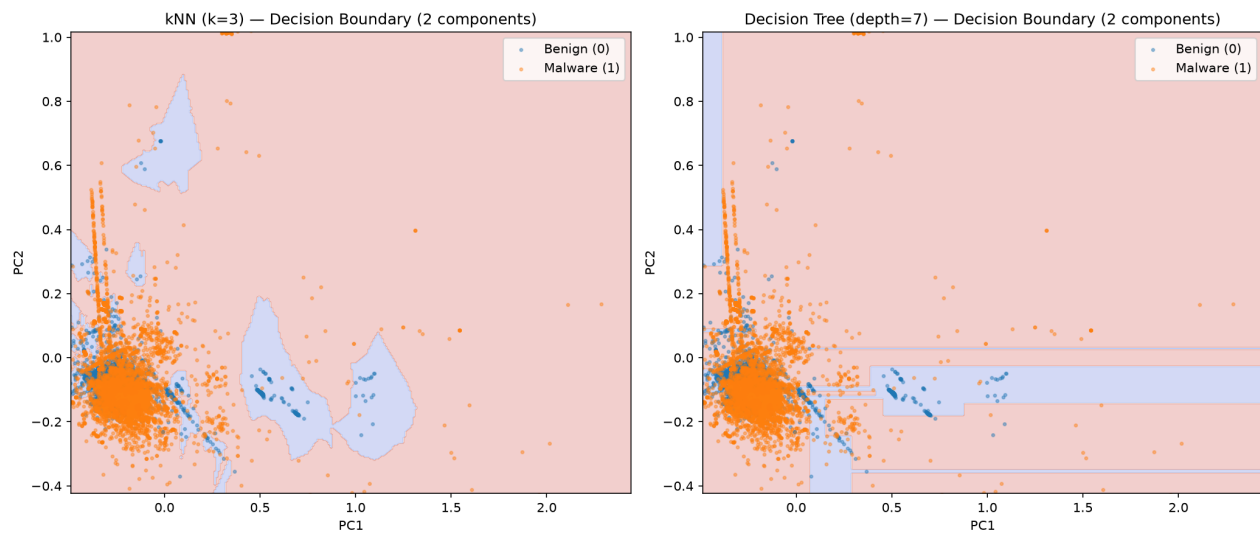


Figure 7. Decision boundaries for both algorithms on the two-component projection. These are illustrative models fitted on 17.6 percent of total variance, not the evaluated classifiers.

The boundary plots explain the performance gap structurally. kNN generates curved, irregular regions that follow local data density, forming several disconnected islands around clusters of benign samples, including a small isolated region enclosing only a few points. The decision tree produces strictly rectangular regions with axis-parallel edges, a direct consequence of each split being a threshold test on a single feature; the tree cannot represent a diagonal boundary at any depth.

The practical cost is visible in the right panel. Where kNN encloses three separate benign clusters with tight contours, the depth-limited tree spans them with a single wide horizontal band, classifying a large region of otherwise empty space as benign. Coarse rectangular regions absorb malicious samples that fall within them, which is consistent with the tree's elevated false negative count.

7. Discussion

7.1 Advantages the metrics do not capture

The decision tree retains properties that Table 3 does not measure. It predicts by traversing at most seven comparisons, whereas kNN must compute distances against all 11,641 training samples for every file classified, a material difference for a scanner processing files at volume. The tree also reports which features drove its decisions, as shown in Figure 4, while kNN offers no comparable account. In a deployment where inference cost or auditability is a binding constraint, those properties could reasonably outweigh a 0.0065 difference in F1.

7.2 Preprocessing relative to model selection

The two most consequential decisions in this analysis involved no model. Removing the leaked identifier prevented a meaningless perfect score, and removing the duplicated header records prevented the same signature from appearing on both sides of the split. Either omission would have produced higher reported metrics and a less valid result. The 0.0065 F1 difference between the two classifiers is small beside the effect either preprocessing error would have introduced.

8. Limitations

Hyperparameter selection used test data. Both k and maximum depth were chosen by comparing test-set performance, which uses the evaluation partition to make modeling decisions and modestly optimizes the reported figures. Cross-validation within the training set would separate the two properly. Given the 0.006 spread across candidate k values, the practical effect here is small, but the methodological point stands.

Dataset composition. Malicious samples derive from a single repository and benign samples appear to originate from standard Windows installations. The class separation observed may therefore be wider than in a deployment encountering software from many vendors and build environments.

Static analysis only. These features describe file structure without execution. Malware engineered to mimic benign header characteristics, or whose malicious behavior manifests only at runtime, would not be distinguished by any model trained on this representation. Static header analysis is one layer of a detection strategy rather than a complete one.

Interpretability cost of PCA. Because components are linear combinations of all 69 input features, the finding that the third component dominates the tree's splits cannot be translated back into a statement about specific PE fields without examining the component loadings.

9. Conclusion

Two classifiers were implemented and compared for static malware detection on 16,631 cleaned PE header records reduced to 30 principal components. kNN with $k = 3$ achieved an F1 of 0.9823 and ROC AUC of 0.9840 against 0.9758 and 0.9639 for a depth-7 decision tree, missing 57 malicious files to the tree's 91. The decision boundary analysis attributes the difference to representational form: kNN's density-following contours enclose the several distinct benign clusters present in this data, while the tree's axis-parallel

rectangles span them coarsely and absorb malicious samples in the process.

Two secondary findings carry beyond the model comparison. First, preprocessing determined the validity of the results more than model choice determined their quality: an unremoved identifier column would have yielded a meaningless perfect score, and unremoved duplicate headers would have inflated both models through memorization. Second, the tree's reliance on the third principal component over the first demonstrates concretely that unsupervised dimensionality reduction orders directions by variance rather than by class relevance, the same limitation that constrains variance-threshold feature selection.