**Data Preparation/Feature Engineering**

**Overview**

The data preparation and feature engineering phase is one of the most critical steps in the Machine Learning model-building process. In our project, we want to classify malware and legitimate software by analyzing extracted features from PE (Portable Executed format), including collecting, cleaning, and comprehending data through Exploratory data analysis, along with feature selection and transformation to improve machine learning models like ExtraTreesClasifier, XGBoost, and Deep Neural Network. Processing this phase impacts the accuracy and reliability of the final result.

**Data Collection**

This project uses a dataset that consists of features extracted from Portable Executable (PE) files belonging to Windows programs - containing metadata, code sections, and structural information. Each record denotes an individual file and consists of some attributes related to file size, the entropy of various sections, import/export table features by extracting by using tools like pefile or similar libraries, and header metadata. The data was loaded from a CSV file named Malware\_Detection\_data.csv, with more than 138,000 samples and 57 features. This file included legitimate columns used as binary labels - 1 for legitimate software and 0 for malware - this served as the target variable for classification tasks.

**Data Cleaning**

In order to ensure the quality and integrity of the data collected for accurate model prediction, cleaning of the data is essential.

* *Missing values*: Checks performed showed no missing values in any of the 57 columns. Therefore, we dropped features with more than 50% missing data, and filled remaining missing values using the median of each column.
* *Outlier Detection*: we used the Interquartile Range (IQR) to detect and optionally fix outliers.
* *Label Balance*: we checked for the skewed distribution of malware vs benign samples and made sure it was reasonable for model training.

**Exploratory Data Analysis (EDA)**

We use EDA to understand data structure and identify patterns and anomalies.

* We plotted the number of malware against benign samples. This would verify whether according to the training results our model would need rebalancing.
* We identified which features are correlated with each other. Features that are closely related are redundant and so can be dropped.
* ExtraTreesClassifier has been applied for ranking all 57 features. For training purposes, only the 13 most important features were selected.

1. SizeOfCode
2. NumberOfSections
3. Entropy
4. SizeOfInitializedData
5. VirtualSize
6. NumberOfImports
7. NumberOfExports
8. ResourcesSize
9. MajorSubsystemVersion
10. DllCharacteristics
11. Characteristics
12. SectionsMeanEntropy
13. ImageBase

* Histograms and boxplots of top features were plotted to analyze their distribution and outliers.

**Feature Engineering**

After the initial cleaning and analysis, we turned to the next thing: a better feature set.

* *Feature Selection:* the 13 most important features were kept using ExtraTreesClassifier because overfitting and noise reduction were sought.
* *Domain-Specific Features*: some features (like SizeOfCode, NumberOfSections, and entropy values) account much for malware behavior with respect to domain knowledge.
* *Reason for Feature Selection-* facilitates models to generalize better by ignoring irrelevant or noisy features. Enhances speed during training and reduces overfitting.

**Data Transformation**

The performance of most machine learning models will improve with good scaling of input features.

*Normalization:*

Min-Max scaling with the MinMaxScaler was performed to scale the features in the range of [0, 1]. The scaling of features holds particular importance for algorithms like Logistic Regression or Neural Networks that are sensitive to how input features are scaled.

*No Categorical Encoding Required:*

There were only numerical features in our dataset, thus no encoding (like one-hot) was required.

*Split Train/Test:*

The classification data sets are split into 70% training and 30% testing while ensuring that the class distributions remain balanced by the use of stratified splitting.

**Model Exploration**

**Model Selection**

Several classifiers mentioned:

|  |  |  |
| --- | --- | --- |
| **Models** | **Strength** | **Weakness** |
| Logistic Regression | Fast and easy to interpret. | Not great at capturing complex, non-linear relationships. |
| Decision Tree | Easy to interpret, handles non-linearity. | Prone to overfitting. |
| Random Forest | Reduces overfitting, high accuracy. | Slower to train than a single tree. |
| XGBoost | Very high predictive performance. | Takes longer to train due to its iterative boosting approach. |
| Deep Neural Networks (DNN) | Can model complex patterns. | Requires large data and longer training time; risk of overfitting with small data. |

The model chosen is: **XGBoost.**

* Reason: XGBoost provides a good balance in terms of the bias-variance tradeoff, it is robust to overfitting, and it has provided the best performance regarding F1-score and accuracy, even though longer training time-wise than simpler models.

**Model Training**

Once we chose XGBoost, the model was configured and trained on cleaned and transformed data.

We used hyperparameters, because

* **n\_estimators = 100:** Increasing the number of trees usually leads to increased accuracy at the expense of increased training time.
* **learning\_rate = 0.1:** It controls how much adjustment is done to the model after each step; a starting value of 0.1 is usually considered a safe option.
* **max\_depth =** 6: A deeper tree will pick up more complexity while remaining shallow enough to avoid overfitting.

Cross Validation: 5-fold Stratified Cross-Validation for training the model on various data subsets for cross-validation, and confirming that performance is stable and does not depend on one random train/test split

**Model Evaluation – How Well Does It Perform**

A wide variety of metrics were used to evaluate the overall performance of the model after training on the unseen test set to get the complete picture.

Metrics Used:

*Accuracy:* How many predictions were anticipated to be correct in general?

*F1 Score:* It can be considered the harmonic mean of Precision and Recall (suitable for imbalanced datasets).

*Confusion Matrix*: A matrix that illustrates the truth against predicted values.

*ROC-AUC Curve:* Indicates how well the model separates classes.

***Key Evaluation Results:***

The accuracy is: 96.5%.

The F1 Score indicates a strong balance between precision and recall.

False Positive: 3.6% of benign software is classified as malicious.

False Negatives: 0.89%, malicious software classified as harmless

These error rates are very low, which indicates that the model is sensitive (it detects malware accurately) and specific (it does not falsely accuse).

Code Implementation – Putting It All Together

# Load data

df = pd.read\_csv("malware\_dataset.csv")

df.fillna(df.median(), inplace=True)

# Select important features using ExtraTreesClassifier

from sklearn.ensemble import ExtraTreesClassifier

selector = ExtraTreesClassifier()

selector.fit(df.drop('label', axis=1), df['label'])

important\_features=df.drop('label', axis=1).columns[selector.feature\_importances\_ > 0.01]

# Normalize the data

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X = scaler.fit\_transform(df[important\_features])

y = df['label']

# Split dataset

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y)

# Train XGBoost

from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X\_train, y\_train)

# Evaluate

from sklearn.metrics import f1\_score

print("F1 Score:", f1\_score(y\_test, model.predict(X\_test)))

# Confusion matrix

ConfusionMatrixDisplay.from\_estimator(model, X\_test, y\_test)