Microsoft Malware Prediction

Can you predict if a machine will soon be hit with malware?

Wenyu Pan

- Problem & Motivation
- Approach
- Implementation
- Evaluation & Results
- Future Directions

Problem & Motivation

Problem:

Predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine.

Motivation:

This predictive analysis is crucial as it enables users and system administrators to take proactive measures to enhance their security. A proactive stance in this domain can significantly reduce disruptions, financial losses, and maintain user trust, thereby contributing to a more secure cyber environment.

Approach

- In my project, I utilized the Microsoft Malware Prediction dataset from Kaggle, applying a unified approach to data processing and modeling.

  DataSet size: DataFrame has 7,853,253 rows and 83 columns.

- I split the data into training and testing sets, then experimented with various models to optimally predict malware occurrences.

Introduction to Microsoft Malware Prediction dataset

HasDetections is the ground truth and indicates that Malware was detected on the machine.

Each row in the dataset corresponds to a machine, uniquely identified by a MachineIdentifier

Q27 | 53

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	MachineIc	ProductNa	EngineVe	AppVersic	AvSigVers	IsBeta	RtpStateB	IsSxsPassi	DefaultBr	AVProduc	AVProduc	AVProduc	HasTpm	CountryId	CityIdenti	Organizati	GeoName	LocaleEng	Platform	Processor	OsVer
2	000002898	win8defe	1.1.15100.	4.18.1807.	1.273.1735	0	7	0		53447	1	1	1	29	128035	18	35	171	windows1	x64	10.0.0.0
3	000007535	win8defe	1.1.14600.	4.13.17134	1.263.48.0	0	7	0		53447	1	1	1	93	1482	18	119	64	windows1	x64	10.0.0.0
4	000007905	win8defe	1.1.15100.	4.18.1807.	1.273.1341	0	7	0		53447	1	1	1	86	153579	18	64	49	windows1	x64	10.0.0.0
5	00000b115	win8defe	1.1.15100.	4.18.1807.	1.273.1527	0	7	0		53447	1	1	1	88	20710		117	115	windows1	x64	10.0.0.0
6	000014a5f	win8defe	1.1.15100.	4.18.1807.	1.273.1379	0	7	0		53447	1	1	1	18	37376		277	75	windows1	x64	10.0.0.0
7	000016191	win8defe	1.1.15100.	4.18.1807.	1.273.1094	0	7	0		53447	1	1	1	97	13598	27	126	124	windows1	x64	10.0.0.0
8	0000161e8	win8defe	1.1.15100.	4.18.1807.	1.273.845.	0	7	0		43927	2	1	1	78	81215		89	88	windows1	x64	10.0.0.0
9	000019515	win8defe	1.1.15100.	4.18.1807.	1.273.1393	0	7	0		53447	1	1	1	97	150323	27	126	124	windows1	x64	10.0.0.0
10	00001a027	win8defe	1.1.15200.	4.18.1807.	1.275.988.	0	7	0		53447	1	1	1	164	155006	27	205	172	windows1	x64	10.0.0.0
11	00001a18c	win8defe	1.1.15100.	4.18.1807.	1.273.973.	0	7	0		46413	2	1	1	93	98572	27	119	64	windows1	x64	10.0.0.0
12	00001b3b3	win8defe	1.1.15100.	4.18.1807.	1.273.869.	0	7	0		53447	1	1	1	107	133897	46	138	134	windows1	arm64	10.0.0.0
13	00001b924	win8defe	1.1.15100.	4.18.1807.	1.273.1826	0	7	0		47238	2	1	1	164	120983	27	205	172	windows1	x64	10.0.0.0
14	00001f26e	win8defe	1.1.15100.	4.18.1807.	1.273.1372	0	7	0		36429	2	1	1	80	7198	27	101	107	windows1	x64	10.0.0.0
15	000024872	win8defe	1.1.15200.	4.18.1807.	1.275.895.	0	7	0		53447	1	1	1	171	124736		211	182	windows1	x64	10.0.0.0
16	0000258d2	win8defe	1.1.15100.	4.18.1807.	1.273.925.	0	7	0		7945	2	1	1	169	141516	27	209	74	windows1	x64	10.0.0.0
17	000027c68	win8defe	1.1.15200.	4.18.1807.	1.275.130.	0	7	0		47238	2	1	1	157	114477	27	199	75	windows1	x64	10.0.0.0
18	00002815C	mse	1.1.15200.	4.9.218.0	1.275.300.	0	7	0		29199	1	1	0	80	7198	27	101	107	windows7	x64	6.1.1.0
19	00002a7fd	win8defe	1.1.15100.	4.18.1806.	1.273.466.	0	0	1		39	1	1	1	93	64168	18	277	75	windows1	x64	10.0.0.0
20	00002b745	win8defe	1.1.15300.	4.18.1809.	1.277.48.0	0	7	0		53447	1	1	1	178	136271	27	230	71	windows1	x64	10.0.0.0
21	00002c6cc	win8defe	1.1.15100.	4.18.1807.	1.273.1795	0	7	0		47238	2	1	1	158	79230	18	202	70	windows1	x64	10.0.0.0
22	0000309dc	win8defe	1.1.15100.	4.10.209.0	1.273.781.	0	7	0		53447	1	1	1	93		27	119	64	windows8	x86	6.3.0.0
23	000033565	win8defe	1.1.15100.	4.8.10240.	1.273.356.	0	7	0		53447	1	1	1	43	12607	27	53	42	windows1	x64	10.0.0.0
24	000037881	win8defe	1.1.15200.	4.18.1807.	1.275.488.	0	7	0		53447	1	1	1	147	77794		187	74	windows1	x64	10.0.0.0
25	000037f84	win8defe	1.1.15200.	4.18.1807.	1.275.173.	0	7	0		7945	2	1	1	12	110781	27	15	58	windows1	x64	10.0.0.0
26	000038f24	win8defe	1.1.15200.	4.18.1806.	1.275.879.	0	7	0		53447	1	1	1	203	143782	27	255	46	windows1	x86	10.0.0.0
27	000039104	win8defe	1.1.15200.	4.18.1807.	1.275.91.0	0	7	0	1950	53447	1	1	1	43	37357	27	53	42	windows1	x86	10.0.0.0
28	000039c28	win8defe	1.1.15100.	4.18.1807.	1.273.1376	0	7	0		53447	1	1	1	205	11382		274	266	windows1	x64	10.0.0.0
29	00003ad63	win8defe	1.1.15300.	4.13.17134	1.277.25.0	0	7	0		53447	1	1	1	171	110673		211	182	windows1	x64	10.0.0.0
30	00003e5e6	win8defe	1.1.15200.	4.18.1807.	1.275.26.0	0	7	0		53447	1	1	1	199	150207	27	266	75	windows1	x64	10.0.0.0
31	0000422df	win8defe	1.1.15100.	4.18.1807.	1.273.1561	0	7	0		53447	1	1	1	9	20805	27	10	214	windows1	x64	10.0.0.0
32	00004348c	win8defe	1.1.15200.	4.18.1807.	1.275.1025	0	7	0		53447	1	1	1	68	59605	27	276	74	windows1	x86	10.0.0.0
33	000043b6c	win8defe	1.1.13504.	4.13.17134	1.237.607.	0	7	0	146	36505	2	1	1	201		27	267	251	windows1	x64	10.0.0.0
34	000046e59	win8defe	1.1.15200.	4.18.1807.	1.275.511.	0	7	0		46669	2	1	1	141	60626	27	240	233	windows1	x64	10.0.0.0

	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	
1	OsBuild	OsSuite	OsPlatfori	OsBuildLa	SkuEditior	IsProtecte	AutoSamp	PuaMode	SMode	IeVerIden	SmartScre	Firewall	UacLuaen	Census_M	Census_D	Census_O	Census_O	Census_Pi	Ce
2	17134	256	rs4	17134.1.ar	Pro	1	0		0	137		1	1	Desktop	Windows.	2668	9124	4	
3	17134	256	rs4	17134.1.ar	Pro	1	0		0	137		1	1	Notebook	Windows.	2668	91656	4	
4	17134	768	rs4	17134.1.ar	Home	1	0		0	137	RequireA	1	1	Desktop	Windows.	4909	317701	4	
5	17134	256	rs4	17134.1.ar	Pro	1	0		0	137	ExistsNot	1	1	Desktop	Windows.	1443	275890	4	
6	17134	768	rs4	17134.1.ar	Home	1	0		0	137	RequireA	1	1	Notebook	Windows.	1443	331929	4	
7	17134	256	rs4	17134.1.ar	Pro	1	0		0	137	RequireA	1	1	Desktop	Windows.	3799	340727	2	
8	17134	768	rs4	17134.1.ar	Home	1	0		0	137		1	1	Notebook	Windows.	3799	207404	2	
9	14393	768	rs1	14393.0.ar	Home	1	0		0	94	RequireA	1	1	Notebook	Windows.	5682	338896	2	
10	17134	256	rs4	17134.1.ar	Pro	1	0		0	137	RequireA	1	1	Notebook	Windows.	2206	240688	4	
11	16299	768	rs3	16299.431	Home	1	0		0		RequireA	1	1	Notebook	Windows.	585	189457	4	
12	17134	256	rs4	17134.1.ar	Pro	1	0		0	137		1	1	Detachabl	Windows.	4143	227191	8	
13	17134	768	rs4	17134.1.ar	Home	1	0		0	137		1	1	Notebook	Windows.	2668	172079	2	
14	17134	256	rs4	17134.1.ar	Pro	1	0		0	137		1	1	Desktop	Windows.	2102	250496	4	
15	17134	256	rs4	17134.1.ar	Pro	1	0		0	137	RequireA	1	1	Desktop	Windows.	4589	313586	2	
16	17134	768	rs4	17134.1.ar	Home	1	0		0	137	Off	1	1	Notebook	Windows.	525	225830	8	
17	17134	256	rs4	17134.1.ar	Pro	1	0		0	137	ExistsNot	1	1	Notebook	Windows.	1443	256567	4	
18	7601	768	windows7	7601.1840	Invalid	1	0		0	290		1	0	Notebook	Windows.	1781	185880	2	
19	16299	256	rs3	16299.431	Pro	1	0		0	117		1	1	Notebook	Windows.	2668	171199	4	
20	17134	768	rs4	17134.1.ar	Home	1	0			137		1	1	Notebook	Windows.	666	264568	4	
21	16299	768	rs3	16299.15.a	Home	1	0		0	111	RequireA	1	1	Notebook	Windows.	2668	171331	4	
22	9600	256	windows8	9600.1906	Pro	1	0		0	333	RequireA	1	1	Notebook	Windows.	2668	35125	2	
23	10240	256	th1	10240.179	Enterprise	1	0		0	65	RequireA	1	1	Notebook	Windows.	666	264571	4	
24	17134	256	rs4	17134.1.ar	Pro	1	0		0	137	RequireA	1	1	Notebook	Windows.	2102	249943	4	
25	17134	768	rs4	17134.1.ar	Home	1	0		0	137	RequireA	1	1	Notebook	Windows.	2653	304618	4	
26	17134	256	rs4	17134.1.x8	Pro	1	0		0	137	ExistsNot	1	1	Desktop	Windows.	585	189698	4	
27	16299	256	rs3	16299.15.x	Pro	1	0		0	117	RequireA	1	1	Desktop	Windows.	4589	313586	2	
28	17134	256	rs4	17134.1.ar	Pro	1	0		0	137		1	1	Desktop	Windows.	2206	251773	4	
29	17134	768	rs4	17134.1.ar	Home	1	0		0	137		1	1	Notebook	Windows.	525	331297	4	
30	17134	256	rs4	17134.1.ar	Pro	1	0		0	137		1	1	PCOther	Windows.	2102	230160	4	
31	16299	256	rs3	16299.15.a	Pro	1	0		0	111		1	1	Desktop	Windows.	1980	227527	8	
32	14393	256	rs1	14393.576	Pro	1	0		0	94	RequireA	1	1	Notebook	Windows.	4730	311910	2	
33	17134	768	rs4	17134.1.ar	Home	0	0		0	137	RequireA	1	1	Notebook	Windows.	2206	244755	4	
34	15063	768	rs2	15063.0.ar	Home	1	0		0	108		1	1	Notebook	Windows.	2668	171476	2	

CE
HasDetect
0
0
0
1
1
1
1
0
0
1
0
1
0
0
1
1
0
0
0
1
0

Before Data Cleaning...EDA

1. The purpose of this EDA process was to understand feature distributions, occurrence frequencies, dimensions, and NaN values, aiding in effective data cleaning.
2. The following steps were done during an EDA”
3. The features, categorized as binary, categorical, and numerical, were analyzed for their maximum, minimum, mean values, standard deviation, and frequency of zeroes.
4. Examining the frequency of occurrences for binary, numerical, and categorical feature types.
5. Assessing the dimensions of each feature.
6. Investigating the frequency of each dimension within the features.
7. Evaluating the prevalence of NaN values across all features.

Data Cleaning

Count NaNs in each feature and find its frequency. We considered NaN frequency over 0.5 as invalid feature and ignore the feature.

1. Deleting features with too much NaN-values

```
nan_count = df.isnull().sum().to_frame('count')
nan_count['count'] = nan_count['count'].div(8921483).round(2)
irrelevant_feature = nan_count[nan_count['count'] > 0.5]
irrelevant_feature
```

```
def assess_balance(df, column):
    value_counts = df[column].value_counts()
    max_count = value_counts.max()
    balance_ratio = max_count / len(df)
    return balance_ratio
```

```
[ ]  unbalanced_df = balance_ratios_df[balance_ratios_df['balance_ratio'] > 0.98]
```

```
change_to_irrelevant(output_df, unbalanced_df)
```

2. Deleting features that are highly unbalanced

Define a function to calculate if the target feature is balanced. Here, we calculate a balance ratio between max count input and total input count. Ratio close to 1 indicates more imbalance.

3.

Features were then separate into three categories:

Numerical (replace NaN values with "-1“)

Categorical (rename NaN-Values as '-1' in all features with tpye 'not category')

Binary(Reassign all NaN-Values to the most fequent feature)

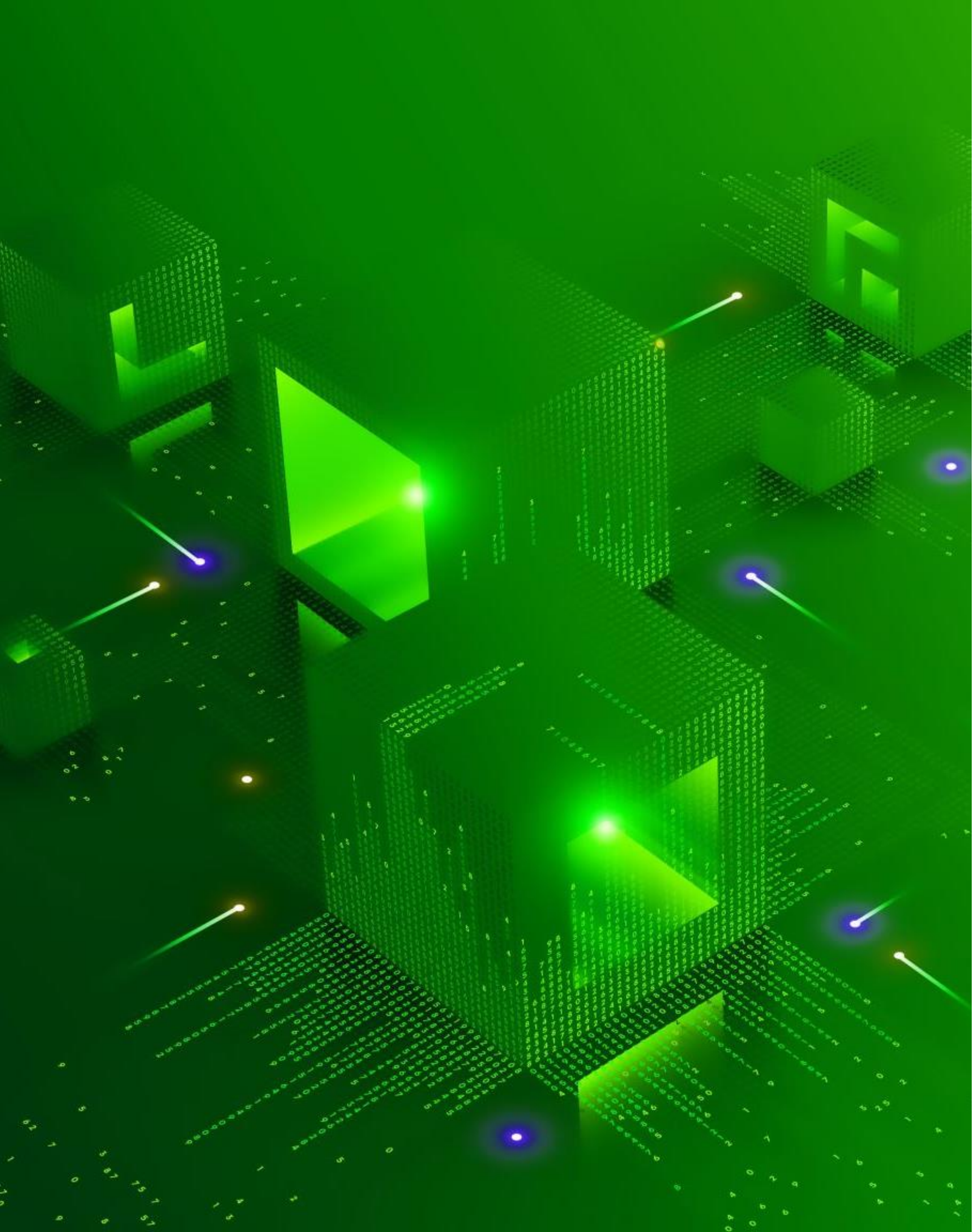
This action ensures numerical features have no missing values, enhancing model accuracy and allowing similar preprocessing for test data.

Now that we are done with data cleaning…

- We move on to data encoding:

1. **Label Encoding**: Transforms categorical values into a numerical range, where each unique label gets a unique integer.
2. **Frequency Encoding**: A variant of Label Encoding where values are encoded based on their frequency, assigning numbers based on the frequency of occurrence.

- This process ensures smooth processing and modeling with machine learning algorithms that typically require numerical input



Implementation

To predict the data, I implemented three model:

- Logistic Regression Model
- Random Forest
- LightGBM/Keras

Logistic Regression Model

Split the dataset into training and testing.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE

# Define your target and data ID columns
target_column = "HasDetections"
data_id = 'MachineIdentifier'


X = df.drop([target_column, data_id], axis=1)
y = df[target_column]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# SMOTE for class imbalance on training data
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

Delete the loaded whole dataset to save memory.

```
[ ] del df
    gc.collect()

    6775
```

Train random forest model.

```
# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)
```

Make prediction on testing dataset and find accuracy.

```
# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print(f"Model Accuracy: {accuracy}")
print(classification_report(y_test, predictions))
```

```
Model Accuracy: 0.5083472510228101
              precision    recall  f1-score   support

           0       0.60      0.05      0.09    222980
           1       0.50      0.97      0.66    223095

    accuracy                           0.51    446075
   macro avg       0.55      0.51      0.38    446075
weighted avg       0.55      0.51      0.38    446075
```

Random Forest

Load encoded dataset

```
[ ] df = pd.read_csv('./drive/MyDrive/train_encoded.csv')
    # df = pd.read_csv('./data/train_encoded.csv')
```

Split the dataset into training and testing.

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Your original columns and data splitting
target_column = "HasDetections"
data_id = 'MachineIdentifier'
X = df.drop([target_column, data_id], axis=1)
y = df[target_column]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE only on training data to handle class imbalance
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

Delete the loaded whole dataset to save memory.

```
[ ] del df
    gc.collect()

    0
```

Train random forest model.

```
[ ] # Initialize the Random Forest model
    model = RandomForestClassifier(random_state=42)  # Use RandomForestRegressor for regression

    # Train the model
    model.fit(X_train, y_train)
```

Make prediction on testing dataset and find accuracy.

```
[ ] # Make predictions
    predictions = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, predictions)
    print(f"Model Accuracy: {accuracy}")

    Model Accuracy: 0.6485097797455585
```

LightGBM

```
[ ] from sklearn.model_selection import train_test_split
    from imblearn.over_sampling import SMOTE

    # Your original columns and data splitting
    target_column = "HasDetections"
    data_id = 'MachineIdentifier'
    X = df.drop([target_column, data_id], axis=1)
    y = df[target_column]

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Apply SMOTE only on training data to handle class imbalance
    sm = SMOTE(random_state=42)
    X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

Delete the loaded whole dataset to save memory.

```
[ ] del df
    gc.collect()

    30
```

Create the LightGBM data containers

```
[ ] train_data = lgb.Dataset(X_train, label=y_train)
    test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)
```

Define the parameters

```
[ ] params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'num_leaves': 31,
        'learning_rate': 0.05,
        'num_iterations': 100
    }
```

Train the model

+ Code + Text

```
[ ] gbm = lgb.train(params, train_data, valid_sets=[test_data])

    /usr/local/lib/python3.10/dist-packages/lightgbm/engine.py:172: UserWarning: Found `num_iterations` in params. Will use it instead of argument
      _log_warning(f"Found `{alias}` in params. Will use it instead of argument")
    [LightGBM] [Info] Number of positive: 892084, number of negative: 892212
    [LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 1.016717 seconds.
    You can set `force_row_wise=true` to remove the overhead.
    And if memory is not enough, you can set `force_col_wise=true`.
    [LightGBM] [Info] Total Bins 5313
    [LightGBM] [Info] Number of data points in the train set: 1784296, number of used features: 66
    [LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499964 -> initscore=-0.000143
    [LightGBM] [Info] Start training from score -0.000143
```

Predict on the test set

```
[ ] y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
```

Convert probabilities to binary output

```
[ ] y_pred_binary = [1 if x > 0.5 else 0 for x in y_pred]
```

Evaluate the model

```
[ ] accuracy = accuracy_score(y_test, y_pred_binary)
    print(f"Accuracy: {accuracy}")

    Accuracy: 0.6446673765622373
```

Evaluation & Results

Model Evaluation Metric:

- Accuracy was used as the primary metric for classification effectiveness.

Logistic Regression Model:

- Achieved 50.83% accuracy.
- Struggled with class imbalances and complex features.

Random Forest Model:

- Reached 64.85% accuracy.
- Benefited from regularization and ensemble methods.

LightGBM and Keras Models:

- LightGBM achieved 64.4% accuracy.
- Keras achieved 63.61% accuracy.
- Comparable performance to Random Forest.
- Advanced techniques showed potential but didn't significantly outperform traditional methods.

Future Directions

Model Evaluation Considerations:

Accuracy alone may not fully capture model performance.

Imbalanced datasets necessitate a cautious interpretation.

Recommended Future Metrics:

Incorporate F1 score, precision, recall, and AUC-ROC.

These metrics provide a more nuanced performance assessment.

Benefits of a Comprehensive Approach:

Facilitates deeper insights into each model's strengths and limitations.

Future Directions for Malware Detection

- **Exploring Additional Data Sources:** Integrating more diverse datasets to cover a wider range of malware signatures and attack vectors.
- **Deep Learning Techniques:** Experimenting with deep neural networks, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), to capture complex patterns in data.
- **Ensemble Methods:** Combining predictions from multiple models to improve accuracy and robustness against diverse malware threats.