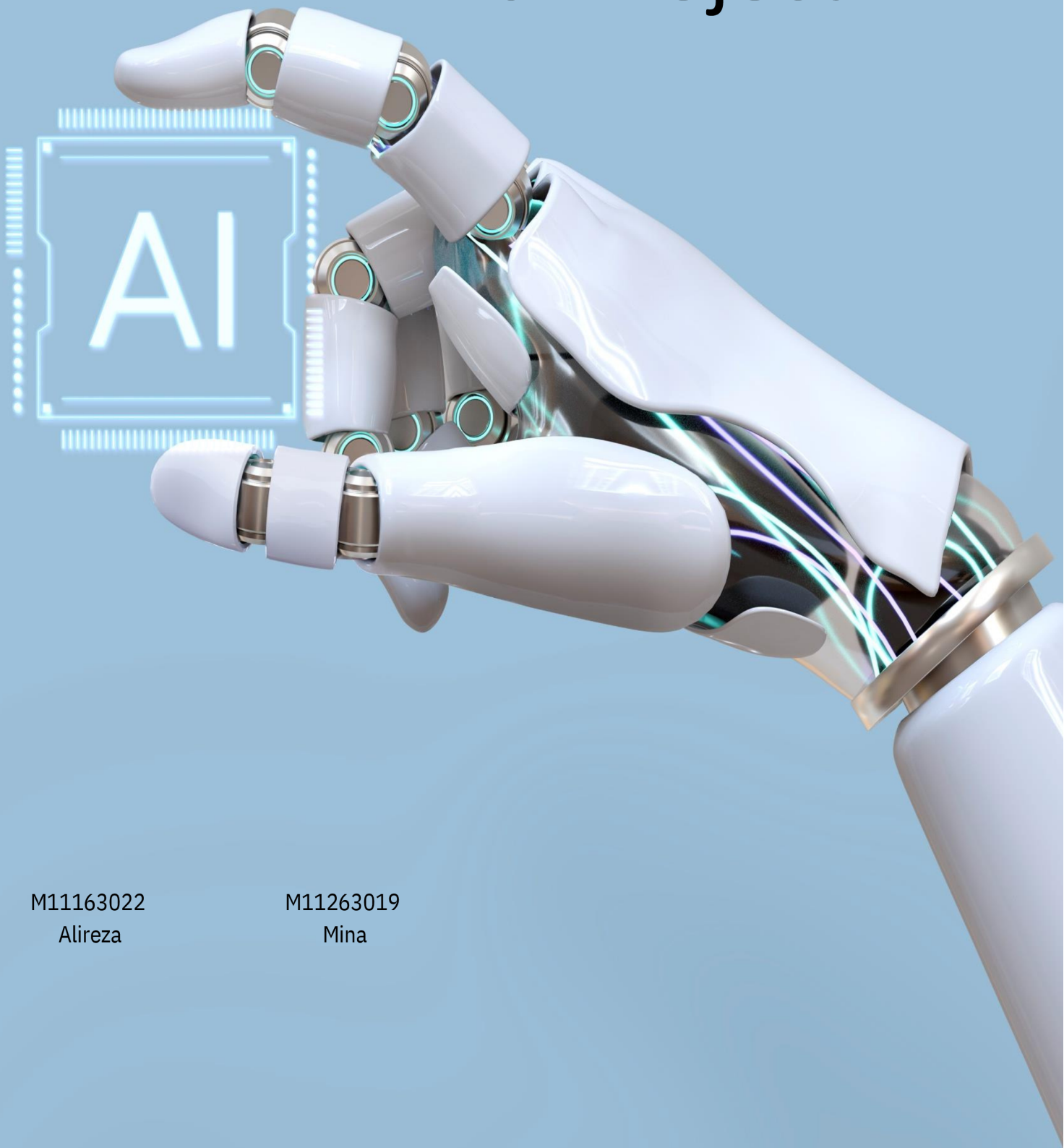
Artificial Intelligence
Final Project



M11163022
Alireza

M11263019
Mina

Early-stage Ransomware Detection based on Pre-Attack Internal API Calls

Introduction:

The article focuses on the growing threat of ransomware attacks, particularly emphasizing the need for early-stage detection. Ransomware is a type of malware designed to encrypt user information or lock access to infected devices, demanding a ransom for the restoration of access. The paper notes the evolving nature of ransomware, with attackers incorporating sensing activities to evade detection by traditional anti- virus and anti-malware solutions. The key problem addressed is the lack of attention to early-stage detection techniques in existing literature. The primary aim of the article is to propose an early-stage ransomware detection system using a neural network model for multi-class classification. The authors seek to detect ransomware before the encryption phase by analyzing pre-attack activities. The proposed model is compared against state-of-the-art approaches, showcasing its superior performance, especially on a challenging, large, and varied dataset that the authors compiled. Additionally, the article aims to contribute to the field by making the dataset, source code, and results publicly available for replication and further research. The dataset used in the study is a critical component of the research. The authors compiled a dataset consisting of 5,203 samples, including 4,753 ransomware samples from 12 different families and 450 benign samples.

Table 2: Ransomware curated dataset

Family	**Samples**
Cerber	450
CryptoWall	450
Matsnu	450
Shade	450
Teslacrypt	450
Benign	450
Hive	443
Ako	432
Erica	377
Conti	359
Matrix	331
Gandcrab	295
Expiro	266
Total	**5,203**

This dataset is highlighted as one of the largest available for ransomware detection. The samples were collected from various online repositories (VirusTotal, Malware Bazaar, VirusShare) spanning the years 2018-2022. The distribution of samples across families is provided, and the authors took steps to ensure a balanced dataset for effective

classification.

OUR WORK:

This research project delves into the realm of artificial intelligence with a focus on supervised learning techniques for classification tasks. The dataset under investigation is labeled, providing a foundation for employing a supervised learning approach. Recognizing the significance of feature extraction, this study employs feature engineering to enhance the predictive capabilities of the chosen machine learning models. In the field of artificial intelligence, the importance of effective feature extraction cannot be overstated. This project aims to explore and implement feature engineering techniques to optimize the performance of supervised learning models on a labeled dataset. Acknowledging the limitations associated with binary datasets, this project endeavors to overcome such constraints through the employment of feature engineering methodologies. Feature engineering serves as a pivotal aspect of this study, addressing the challenges posed by binary datasets. The conventional wisdom of binary datasets being suboptimal is acknowledged, and as a countermeasure, feature engineering techniques are implemented to extract relevant information and improve the discriminatory power of the models.

Table 3: Evasion APIs

Category	Evasion techniques	Evasion API	Description
Data access and storage	Unpacking	MoveFileWithProgressW	Move a file or directory, including its children
	Environment Sensing	NtCreateFile	Creates a new file or directory or opens an existing file
	Process Injection	NtWriteFile	Write data to an open file
		SetFileAttributesW	Sets the attributes for a file or directory
		GetDiskFreeSpaceExW	Retrieve information about the amount of space available on a disk
		GetDiskFreeSpaceW	Retrieves information about the specified disk
		ShellExecuteExW	Perform an operation on a specified file
		DeviceIoControl	Send a control code directly to a specified device driver
Generic OS queries	Environment Sensing	GetComputerNameW	Retrieve the name of the local computer
		NtQuerySystemInformation	Retrieve the specified system information
Memory management	Unpacking	GlobalMemoryStatusEx	Retrieve information about the system memory usage
	Environment Sensing	NtAllocateVirtualMemory	Reserve a region of pages within the user-mode virtual address space
	Process Injection	NtMapViewOfSection	Map specified part of Section Object into process memory
		NtProtectVirtualMemory	Change the protection on a region of committed pages
		NtUnmapViewOfSection	Unmap a view of a section from the virtual address space
		WriteProcessMemory	Writes data to an area of memory in a specified process
		LdrGetDllHandle	Loads a file in memory
Network	Unpacking	GetAdaptersAddresses	Retrieve the addresses associated with the adapters
	Environment Sensing	InternetOpenA	Initialize an application's use of the WinINet functions
Process	Process Injection	CreateProcessInternalW	Create a new process and its primary thread
		NtGetContextThread	Return the user-mode context of the specified thread
		NtResumeThread	Map specified part of Section Object into process memory
		NtSetContextThread	Set the user-mode context of the specified thread
		NtTerminateProcess	Terminate a process and all of its threads
		Process32NextW	Retrieve information about the next process recorded in a snapshot
		NtLoadDriver	Load a driver into the system
Registry	Process Injection	NtSetValueKey	Create or replaces a registry key's value entry
	Environment Sensing	RegOpenKeyExW	Open the specified registry key
		RegQueryValueExW	Retrieve the type and data for the specified value name of a key
		RegSetValueExW	Set the data and type of a specified value under a registry key
		NtCreateKey	Create a new registry key or opens an existing one
Security	Process Injection	CryptGenKey	Generate a random cryptographic session key or a key pair
		CryptExportKey	Export a cryptographic key or a key pair
		LookupPrivilegeValueW	Retrieve the identifier used to represent the specified privilege name
		CryptHashData	Add data to a specified hash object
Services	Environment Sensing	CreateServiceW	Create a service object and adds it to the specified service manager
		EnumServicesStatusW	Enumerate services in the specified service control manager database
UI artifacts	Environment Sensing	SetWindowsHookExW	Install an application-defined hook procedure into a hook chain
		FindWindowW	Retrieve a handle to the top-level window

Feature engineering plays a crucial role in refining the dataset and extracting meaningful insights to improve the performance of machine learning models. In this study, a thoughtful approach to feature engineering was adopted, specifically focusing on enhancing the representation of various aspects of the dataset. To achieve this, the features related to Data Access and Storage were enriched by aggregating information from eight distinct Evasion APIs. The process involved summing the occurrences of these APIs within the Data Access and Storage category and then normalizing by dividing the sum by the total number of rows. This transformation not only consolidates relevant information but also ensures that the engineered feature provides a normalized perspective, enabling the machine learning models to discern patterns and relationships more effectively within the Data Access and Storage domain. Furthermore, this feature engineering methodology was extended across multiple categories, including Generic OS Queries, Memory Management, Network, Process, Registry, Security, Services, and UI Artifacts. Each category underwent a similar process of aggregation, summing, and normalization, thereby creating a set of enriched features that capture the essence of the original dataset more comprehensively. This meticulous feature engineering approach adds a layer of depth to the dataset, enriching it with higher-level abstractions that can potentially contribute to improved model generalization and predictive accuracy across a diverse set of AI applications and scenarios. The chosen methodology involves the application of three distinct machine learning models: Artificial Neural Networks (ANN), Decision Trees, and ensemble methods including XGBoost and Random Forest. Each model is systematically trained and evaluated on the labeled dataset, with a keen emphasis on the feature engineering processes incorporated into the pipeline.

Machine Learning Models:

1. Artificial Neural Networks (ANN): Artificial Neural Networks (ANN) deep exploration of the dataset's intricate patterns. The neural network architecture is designed to learn complex relationships within the feature space, thereby enhancing the model's ability to make accurate predictions.
2. Decision Trees: Decision trees provide a transparent and interpretable framework for classification. The study employs decision trees to analyze and partition the feature space, creating a hierarchical structure that aids in the discernment of decision boundaries.
3. Ensemble Methods (XGBoost and Random Forest): Ensemble methods, namely XGBoost and Random Forest, are employed to harness the collective

wisdom of multiple weak learners. These models are known for their robustness and effectiveness in handling complex datasets.

Here is what is going on inside the scripts:
Importing part:

Import modules

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

#ML auxiliary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
#SkLearn auxiliary libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, ConfusionMatrixDisplay, f1_score, precision_score, recall_score, top_k_accuracy_score
#SkLearn classifiers
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import seaborn as sns
```

Import datasets

```
our_dataset = pd.read_csv('Mydataset.csv')
data = our_dataset.to_numpy()
x_ours = data[:, :-1].astype(float)
y_ours = our_dataset['Family'].values
x_train_ours, x_test_ours, y_train_ours, y_test_ours = train_test_split(x_ours, y_ours, test_size=0.2)
# Assuming x_ours and y_ours are your input features and labels, respectively.
unique_classes = np.unique(np.concatenate([y_train_ours, y_test_ours]))
class_mapping = {cls: idx for idx, cls in enumerate(unique_classes)}
y_train_ours = np.array([class_mapping[cls] for cls in y_train_ours])
y_test_ours = np.array([class_mapping[cls] for cls in y_test_ours])
y_ours = np.array([class_mapping[cls] for cls in y_ours])
```

Results:

Random Forest:

Random Forest

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, top_k_accuracy_score, confusion_matrix

# Assuming you have x_train_ours, y_train_ours, x_test_ours, y_test_ours

# Create RandomForestClassifier
rand_for_classifier = RandomForestClassifier()

# Perform cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  # Adjust the number of splits as needed
cv_scores = cross_val_score(rand_for_classifier, x_ours, y_ours, cv=cv, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())
```

```
Cross-Validation Scores: [0.97790586 0.97790586 0.97598463 0.98076923 0.97692308]
Mean CV Accuracy: 0.9778977314712185
```

XGBoost:

```
!pip install xgboost

import xgboost as xgb
from sklearn.model_selection import cross_val_score

# Assuming x_ours and y_ours are your input features and labels, respectively.

# Split the data into training and testing sets

# Define XGBoost parameters
xgb_params = {
    'colsample_bytree': 0.9,
    'learning_rate': 0.1,
    'max_depth': 9,
    'n_estimators': 200,
    'subsample': 0.9,
    'objective': 'multi:softmax',
    'num_class': len(set(y_train_ours)),
    'eval_metric': 'mlogloss'
}

# Create XGBoost classifier
xgb_model = xgb.XGBClassifier(**xgb_params)

# Use cross-validation to evaluate the model
cv_scores = cross_val_score(xgb_model, x_ours, y_ours, cv=5, scoring='accuracy')
print("Cross-validation accuracy:", cv_scores)

print("Cross-validation mean accuracy:", cv_scores.mean())
```

```
[notice] A new release of pip is available: 23.3.1 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: xgboost in c:\users\user\anaconda3\lib\site-packages (2.0.2)
Requirement already satisfied: numpy in c:\users\user\appdata\roaming\python\python311\site-packages (from xgboost) (1.26.1)
Requirement already satisfied: scipy in c:\users\user\appdata\roaming\python\python311\site-packages (from xgboost) (1.11.4)
Cross-validation accuracy: [1. 1. 1. 1. 1.]
Cross-validation mean accuracy: 1.0
```

LGBMClassifier

```
from lightgbm import LGBMClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import accuracy_score

# Assuming x_ours and y_ours are your input features and labels, respectively.

# Split the data into training and testing sets
x_train_ours, x_test_ours, y_train_ours, y_test_ours = train_test_split(x_ours, y_ours, test_size=0.2, random_state=42)

# Create the LightGBM classifier with adjusted parameters
lgbm_model = LGBMClassifier(
    max_bin=500,
    learning_rate=0.05,
    num_iterations=1000,
    num_leaves=50,  # Adjust based on the size of your dataset and complexity
    boosting_type='dart',
    categorical_feature=[0, 1, 2]  # Specify categorical features indices
)

# Use cross-validation to evaluate the model
cv_scores = cross_val_score(lgbm_model, x_ours, y_ours, cv=5, scoring='accuracy')
print("Cross-validation accuracy:", cv_scores)
print("Cross-validation mean accuracy:", cv_scores.mean())
```

```
Cross-validation accuracy: [1. 1. 1. 1. 1.]
Cross-validation mean accuracy: 1.0
```

KNN:

K-Nearest Neighbors

```
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, top_k_accuracy_score

# Assuming x_ours and y_ours are your full dataset

# Split the data into training and testing sets
x_train_ours, x_test_ours, y_train_ours, y_test_ours = train_test_split(x_ours, y_ours, test_size=0.2, random_state=42)

# Initialize KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)

# Perform cross-validation on the training set
cv_scores = cross_val_score(knn, x_train_ours, y_train_ours, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Cross-validation mean score:", cv_scores.mean())

# Fit the model on the training data
```

```
Cross-Validation Scores: [0.95318127 0.96518607 0.96033654 0.94831731 0.96153846]
Cross-validation mean score: 0.9577119309262165
```

MLP:

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Assuming x_ours and y_ours are your full dataset

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_ours, y_ours, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Initialize MLPClassifier with hidden layers
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, random_state=42)

# Fit the model on the training data
mlp_classifier.fit(x_train_scaled, y_train)

# Make predictions on the test data
y_pred = mlp_classifier.predict(x_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("MLP Classifier Accuracy:", accuracy)
```

```
MLP Classifier Accuracy: 0.9750240153698367
```

ANN:

ANN

```
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.metrics import TopKCategoricalAccuracy
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, precision_score, recall_score, f1_score, accuracy_score

# Assuming x_ours and y_ours are your input features and labels, respectively.

# Add the following import statement for to_categorical
from keras.utils import to_categorical

y_cat_ours = to_categorical(y_ours)
x_train_ours, x_test_ours, y_train_ours, y_test_ours = train_test_split(x_ours, y_cat_ours, test_size=0.2)

# Update input_dim to match the number of features in your input data
model = Sequential()
model.add(Dense(512, input_dim=74, activation="relu"))  # Update input_dim to match your data
model.add(Dense(256, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(13, activation="softmax"))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy', TopKCategoricalAccuracy(k=2)])
model.summary()
model.fit(x_train_ours, y_train_ours, verbose=1, epochs=100, batch_size=10)

predict_x = model.predict(x_test_ours)
y_pred_class = np.argmax(predict_x, axis=1)

y_pred = model.predict(x_test_ours)
y_test_class = np.argmax(y_test_ours, axis=1)
print(confusion_matrix(y_test_class, y_pred_class))

print(classification_report(y_test_class, y_pred_class))

print('precision: ' + str(precision_score(y_pred_class, y_test_class, average='weighted')))
print('recall: ' + str(recall_score(y_pred_class, y_test_class, average='weighted')))
print('f1-score: ' + str(f1_score(y_pred_class, y_test_class, average='weighted')))
print('accuracy: ' + str(accuracy_score(y_pred_class, y_test_class)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        85
           1       0.95      0.97      0.96        77
           2       0.97      0.96      0.97        78
           3       0.98      0.99      0.98        84
           4       0.99      0.99      0.99        83
           5       0.97      1.00      0.99        36
           6       1.00      1.00      1.00        54
           7       1.00      0.99      0.99        99
           8       1.00      0.91      0.95        68
           9       0.97      1.00      0.98        98
          10       0.98      0.99      0.98        99
          11       0.97      0.99      0.98        88
          12       1.00      1.00      1.00        92

    accuracy                           0.98      1041
   macro avg       0.98      0.98      0.98      1041
weighted avg       0.98      0.98      0.98      1041

precision: 0.9832164765034743
recall: 0.9827089337175793
f1-score: 0.982768707153347
accuracy: 0.9827089337175793
```

Here you can see the comparison of the results of the authors with different models in different metrics:

Table 5: Multi-class classification results

Model	Precision	Recall	F1-Score	Accuracy	Top-k Acc. (k=2)
Random Forest	81.23%	78.38%	78.28%	78.38%	85.82%
Bernoulli Naïve Bayes	61.41%	56.38%	55.94%	56.38%	67.33%
K-Nearest Neighbors	78.39%	75.98%	76.07%	75.98%	82.03%
Artificial Neural Network	82.00%	80.00%	81.00%	**80.00%**	**90.41%**

To compare our results with the authors we provided the following table:

Work	Random Forest	KNN	MLP	ANN
Authors (Acuuracy)	78.38%	75.98%	----------	80%
Our (Accuracy)	97.79%	95.77%	97.50%	98.27%

P.S.: The numbers in bold shows the better result.

It should be noted that the feature engineered dataset by this work is attached with the final ZIP file.