Data Preparation/Feature Engineering

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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

1. 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.

2. 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

3. 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 – Snapshots

Step1: Collecting Data

```
#import necesary library
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import os
import pickle
import pefile
import joblib
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
%matplotlib inline
```

```
#load the dataset
df = pd.read_csv("Malware_Detection_data.csv",sep="|",low_memory=True)
df
```


```
#load the dataset
df = pd.read_csv("Malware_Detection_data.csv",sep="|",low_memory=True)
df
```


	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLi
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	
...	...	...	...	...	...	...	
138042	VirusShare_8e292b418568d6e7b87f2a32aee7074b	8e292b418568d6e7b87f2a32aee7074b	332	224	258	11	
138043	VirusShare_260d9e2258aed4c8a3bbd703ec895822	260d9e2258aed4c8a3bbd703ec895822	332	224	33167	2	
138044	VirusShare_8d088a51b7d225c9f5d11d239791ec3f	8d088a51b7d225c9f5d11d239791ec3f	332	224	258	10	
138045	VirusShare_4286dccf67ca220fe67635388229a9f3	4286dccf67ca220fe67635388229a9f3	332	224	33166	2	
138046	VirusShare_d7648eae45f09b3adb75127f43be6d11	d7648eae45f09b3adb75127f43be6d11	332	224	258	11	

138047 rows × 57 columns

Step2: Preparing the Data

2.1: Visualize the data

```
df.shape
```

```
(138047, 57)
```

```
df.describe()
```

	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfUninitializedData	Addre:
count	138047.000000	138047.000000	138047.000000	138047.000000	138047.000000	1.380470e+05	1.380470e+05	1.380470e+05	
mean	4259.069274	225.845632	4444.145994	8.619774	3.819286	2.425956e+05	4.504867e+05	1.009525e+05	
std	10880.347245	5.121399	8186.782524	4.088757	11.862675	5.754485e+06	2.101599e+07	1.635288e+07	
min	332.000000	224.000000	2.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	
25%	332.000000	224.000000	258.000000	8.000000	0.000000	3.020800e+04	2.457600e+04	0.000000e+00	
50%	332.000000	224.000000	258.000000	9.000000	0.000000	1.136640e+05	2.631680e+05	0.000000e+00	
75%	332.000000	224.000000	8226.000000	10.000000	0.000000	1.203200e+05	3.850240e+05	0.000000e+00	
max	34404.000000	352.000000	49551.000000	255.000000	255.000000	1.818587e+09	4.294966e+09	4.294941e+09	

8 rows × 55 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 138047 entries, 0 to 138046
Data columns (total 57 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Name                         138047 non-null  object
 1   md5                          138047 non-null  object
 2   Machine                      138047 non-null  int64
 3   SizeOfOptionalHeader         138047 non-null  int64
 4   Characteristics              138047 non-null  int64
 5   MajorLinkerVersion           138047 non-null  int64
 6   MinorLinkerVersion           138047 non-null  int64
 7   SizeOfCode                   138047 non-null  int64
 8   SizeOfInitializedData        138047 non-null  int64
 9   SizeOfUninitializedData      138047 non-null  int64
 10  AddressOfEntryPoint          138047 non-null  int64
 11  BaseOfCode                   138047 non-null  int64
 12  BaseOfData                   138047 non-null  int64
```

```
df['legitimate'].value_counts()
```

```
0    96724
1    41323
Name: legitimate, dtype: int64
```

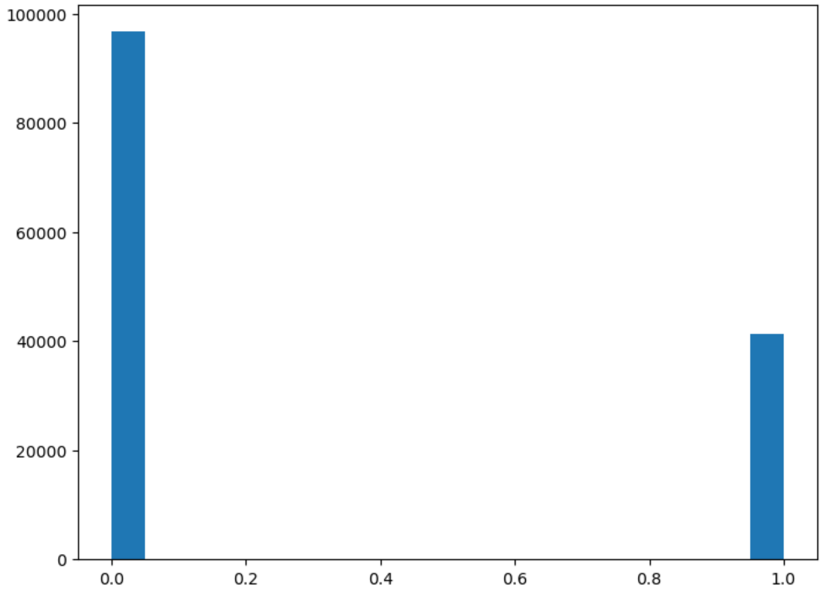
```python
legitimet_data = df[0:41323].drop(["legitimate"],axis=1)
malware_data = df[41323::].drop(['legitimate'],axis=1)

print("shape of legititimet dataset is: %s samples, %s features"%(legitimet_data.shape[0],
                                                                    legitimet_data.shape[1]))


print("shape of malware dataset is: %s samples, %s features"%(malware_data.shape[0],
                                                                malware_data.shape[1]))
```

```
shape of legititimet dataset is: 41323 samples, 56 features
shape of malware dataset is: 96724 samples, 56 features
```

```python
malware_data.columns
```

```
Index(['Name', 'md5', 'Machine', 'SizeOfOptionalHeader', 'Characteristics',
       'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode',
       'SizeOfInitializedData', 'SizeOfUninitializedData',
       'AddressOfEntryPoint', 'BaseOfCode', 'BaseOfData', 'ImageBase',
       'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion',
       'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion',
       'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfImage',
       'SizeOfHeaders', 'CheckSum', 'Subsystem', 'DllCharacteristics',
       'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve',
       'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'SectionsNb',
       'SectionsMeanEntropy', 'SectionsMinEntropy', 'SectionsMaxEntropy',
       'SectionsMeanRawsize', 'SectionsMinRawsize', 'SectionMaxRawsize',
       'SectionsMeanVirtualsize', 'SectionsMinVirtualsize',
       'SectionMaxVirtualsize', 'ImportsNbDLL', 'ImportsNb',
       'ImportsNbOrdinal', 'ExportNb', 'ResourcesNb', 'ResourcesMeanEntropy',
       'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'ResourcesMeanSize',
       'ResourcesMinSize', 'ResourcesMaxSize', 'LoadConfigurationSize',
       'VersionInformationSize'],
      dtype='object')
```

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])  #add_axes(x0, y0, width, height)
ax.hist(df["legitimate"],20)
plt.show()
```



2.2: Data Wrangling

```
df.isnull().sum()
```

```
Name                            0
md5                             0
Machine                         0
SizeOfOptionalHeader            0
Characteristics                 0
MajorLinkerVersion              0
MinorLinkerVersion              0
SizeOfCode                      0
SizeOfInitializedData           0
SizeOfUninitializedData         0
AddressOfEntryPoint             0
BaseOfCode                      0
BaseOfData                      0
ImageBase                       0
SectionAlignment                0
FileAlignment                   0
MajorOperatingSystemVersion     0
MinorOperatingSystemVersion     0
MajorImageVersion               0
MinorImageVersion               0
MajorSubsystemVersion           0
MinorSubsystemVersion           0
SizeOfImage                     0
SizeOfHeaders                   0
CheckSum                        0
Subsystem                       0
DllCharacteristics              0
SizeOfStackReserve              0
SizeOfStackCommit               0
SizeOfHeapReserve               0
SizeOfHeapCommit                0
LoaderFlags                     0
NumberOfRvaAndSizes             0
SectionsNb                      0
SectionsMeanEntropy             0
SectionsMinEntropy              0
SectionsMaxEntropy              0
SectionsMeanRawsize             0
SectionsMinRawsize              0
SectionMaxRawsize               0
SectionsMeanVirtualsize         0
SectionsMinVirtualsize          0
SectionMaxVirtualsize           0
```

2.3: Train test split

```
#spliting the data using train_test_split() methode
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.2, random_state=42)
```

```
X_train.shape
```

```
(110437, 54)
```

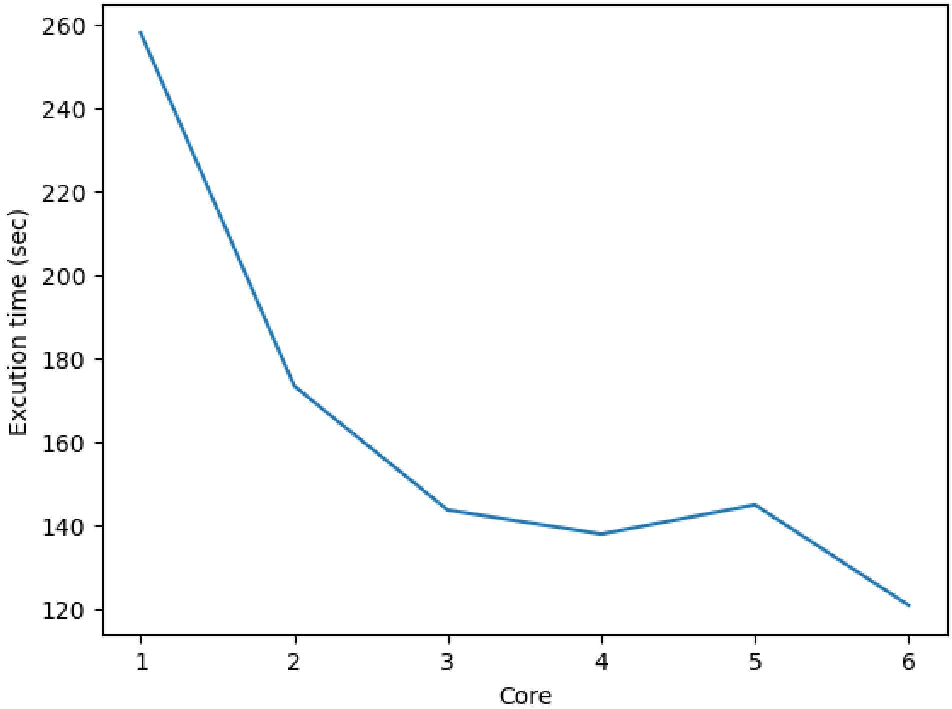
Step3: Choosing, Training and Evaluating a Model

A. Random Forest:

```
#Choosing random forest model
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import f1_score,accuracy_score,plot_confusion_matrix,auc,confusion_matrix
from time import time
```

```
results = list()
# compare timing for number of cores
n_cores = [1, 2, 3, 4, 5, 6]
for n in n_cores:
    # capture current time
    start = time()
    # define the model
    model = RandomForestClassifier(n_estimators=500, n_jobs=n)
    # fit the model
    randomModel = model.fit(X_train, y_train)

    # capture current time
    end = time()
    # store execution time
    result = end - start
    print('>cores=%d: %.3f seconds' % (n, result))

    results.append(result)
plt.ylabel("Excution time (sec)")
plt.xlabel("Core")
plt.plot(n_cores, results)
plt.show()
```

```
>cores=1: 257.887 seconds
>cores=2: 173.334 seconds
>cores=3: 143.651 seconds
>cores=4: 137.908 seconds
>cores=5: 144.910 seconds
>cores=6: 120.854 seconds
```




```python
clf = RandomForestClassifier(max_depth = 2, random_state = 42, n_jobs = -2 )
#training a model
start_time = time()
randomModel = clf.fit(X_train, y_train)

#testing a model
train_pred=randomModel.predict(X_train)
prediction = randomModel.predict(X_test)

#model evaluation
acc_score_tr = accuracy_score(train_pred, y_train)
acc_score_ts = accuracy_score(y_test,prediction)
print("Accuracy: Accuracy on training dataset is %.2f%% and testing dataset is %.2f%% "%(acc_score_tr*100.0, acc_score_ts*100.0))
f1_s = f1_score(y_test, prediction)
print("F1_Score: %.2f%%" %(f1_s*100.0) )

end_time = time()
Excution_time = end_time - start_time
print("Excution time was {}secs".format(Excution_time) )
```

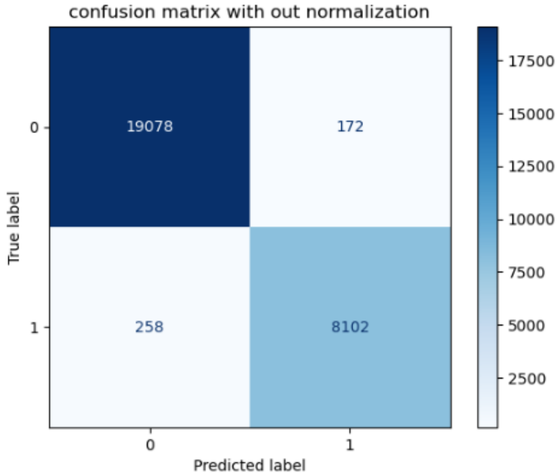
```
Accuracy: Accuracy on training dataset is 98.35% and testing dataset is 98.44%
F1_Score: 97.41%
Excution time was 8.528741598129272secs
```

```
#confusion matrix
conf_matrix = confusion_matrix(y_test,prediction)
conf_matrix
```

```
array([[19078,   172],
       [  258,  8102]], dtype=int64)
```

```
titles_options = [("confusion matrix with out normalization",None),
                 ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(randomModel, X_test, y_test,

                                cmap=plt.cm.Blues,
                                normalize = normalize)
    disp.ax_.set_title(title)
    print(title)
    plt.show()
```

```
C:\Users\Getcho\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
confusion matrix with out normalization
```



B. XGBoost classifier:

```
from xgboost import XGBClassifier
```

```
xgb1 = XGBClassifier(n_estimators = 42, n_jobs=-2, random_state = 4)
start_time = time()
xgb1.fit(X_train, y_train)

# make predictions for training and testing dataset
y_pred = xgb1.predict(X_test)
tra_predict = xgb1.predict(X_train)
pred = [round(values) for values in tra_predict]
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy_ts = accuracy_score(y_test, predictions)
accuracy_tr = accuracy_score(y_train, pred)

print("Accuracy: training Dataset is %.2f%% and testing dataset is %.2f%%" % (accuracy_tr * 100.0, accuracy_ts*100.0))

end_time = time()
Excution_time = end_time - start_time
print("Excution time was {}secs".format(Excution_time) )
```

```
Accuracy: training Dataset is 99.75% and testing dataset is 99.46%
Excution time was 18.150291681289673secs
```

```
from sklearn.metrics import mean_squared_error
msq = mean_squared_error(y_test, y_pred)
print("Mean square error: %.2f%% "%(msq*100.0))
```

```
Mean square error: 0.54%
```

C. Logistic Regression:

```
#Choosing a model
from sklearn.linear_model import LogisticRegression
```

```
clf = LogisticRegression( multi_class='auto', verbose=0,n_jobs = -2)
#train a model
start_time = time()
logModel = clf.fit(X_train, y_train)

#accuracy on training and testing dataset
train_log=logModel.predict(X_train)
pred = logModel.predict(X_test)
acc_tr = accuracy_score(y_train,train_log)
acc_ts = accuracy_score(y_test,pred)
print("Accuracy on training dataset is %.2f%% and on testing dataset is %.2f%% "%(acc_tr*100.0, acc_ts*100.0))
f1 = f1_score(y_test, pred)
print("F1_score: %.2f%% "%(f1*100.0))

end_time = time()
Excution_time = end_time - start_time
print("Excution time was {}secs".format(Excution_time) )
```

```
Accuracy on training dataset is 70.15% and on testing dataset is 69.72%
F1_score: 0.00%
Excution time was 111.32745933532715secs
```

C. Neural Network:

```
#Choosing a model
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense
```

```
# modele = keras.Sequential()
# modele.add(Dense(64, input_dim = 54, activation='relu'))
# modele.add(Dense(16, activation='relu'))
# modele.add(Dense(4,activation='relu'))
# modele.add(Dense(1, activation='sigmoid'))

# modele.summary()
```

```
modele= keras.Sequential([
                   Dense(16, input_dim = 54, activation='relu'),
                   Dense(8, activation='relu'),
                   Dense(4,activation='relu'),
                   Dense(1, activation='sigmoid')
                  ])
modele.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                880

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 4)                 36

 dense_3 (Dense)             (None, 1)                 5

=================================================================
Total params: 1,057
Trainable params: 1,057
Non-trainable params: 0
_________________________________________________________________
```

```python
#compile the model
modele.compile(loss='binary_crossentropy',optimizer = 'rmsprop', metrics=['accuracy'])
```

```python
#train a model
modele.fit(X_train, y_train, epochs=5, batch_size=32)
```

```
Epoch 1/5
3452/3452 [==============================] - 13s 4ms/step - loss: 0.6090 - accuracy: 0.7015
Epoch 2/5
3452/3452 [==============================] - 13s 4ms/step - loss: 0.6091 - accuracy: 0.7015
Epoch 3/5
3452/3452 [==============================] - 12s 4ms/step - loss: 0.6091 - accuracy: 0.7015
Epoch 4/5
3452/3452 [==============================] - 12s 4ms/step - loss: 0.6090 - accuracy: 0.7015
Epoch 5/5
3452/3452 [==============================] - 13s 4ms/step - loss: 0.6090 - accuracy: 0.7015
```

```
<keras.callbacks.History at 0x2b79362cee0>
```

```python
#model evaluation
#accuracy on training dataset
trainPred = modele.predict(X_train)
trainPred = [1 if y>=0.5 else 0 for y in trainPred]
accuracy_score(y_train, trainPred)
```

```
3452/3452 [==============================] - 9s 3ms/step
```

```
0.7015221347917817
```

```python
#accuracy on test dataset
y_prediction = modele.predict(X_test)
y_prediction = [1 if y>=0.5 else 0 for y in y_prediction]
accuracy_score(y_test, y_prediction)
```

```
863/863 [==============================] - 2s 2ms/step
```

```
0.6972111553784861
```

```python
#confusion matrix
conf_matrix_neural = confusion_matrix(y_test, y_prediction)
conf_matrix_neural
```

```
array([[19250,     0],
       [ 8360,     0]], dtype=int64)
```