Importing Modules

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
```

Loading Dataset

```
#Viene caricato il dataset
size = 5000
dataframe = pd.read_csv("../../Datasets/"+str(size)+"dataset.csv")
dataframe
```

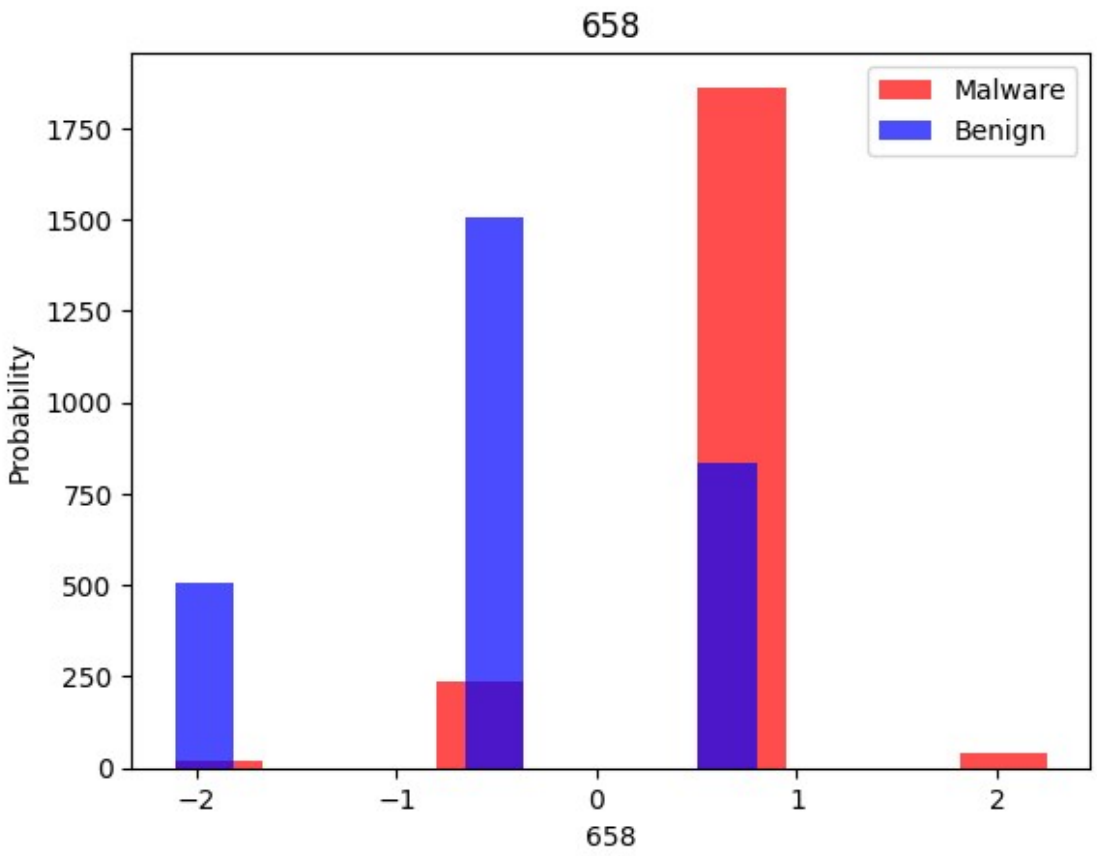
```
      is_malicious       658      2359      2360       613
930  \
0                1 -0.654240 -0.260646 -0.143561 -0.198090  1.157200

1                1  0.798335 -0.260646 -0.143561 -0.215691 -0.454051

2                1  0.798335 -0.260646 -0.143561 -0.215691  1.157200

3                0 -0.654240  0.793149 -0.139496  0.382767 -0.454051

4                1  0.798335 -0.260646 -0.143561 -0.215691  1.157200

...            ...       ...       ...       ...       ...       ...

4995             0  0.798335  0.164560  0.560006 -0.110081 -0.454051

4996             0  0.798335 -0.260646 -0.143561 -0.215691  1.157200

4997             0 -2.106814 -0.260646 -0.143561 -0.215691 -0.454051

4998             0 -0.654240  0.685870 -0.143249  0.277156 -0.454051

4999             1  0.798335 -0.260646 -0.143561  0.294758  1.157200


           637       623       642      2363  ...       698       798
\
0     0.558562 -0.534797  1.102865 -0.048695  ...  0.060938  0.101269

1     0.558562 -0.534797 -0.906729 -0.048695  ...  0.061939  0.145654
```

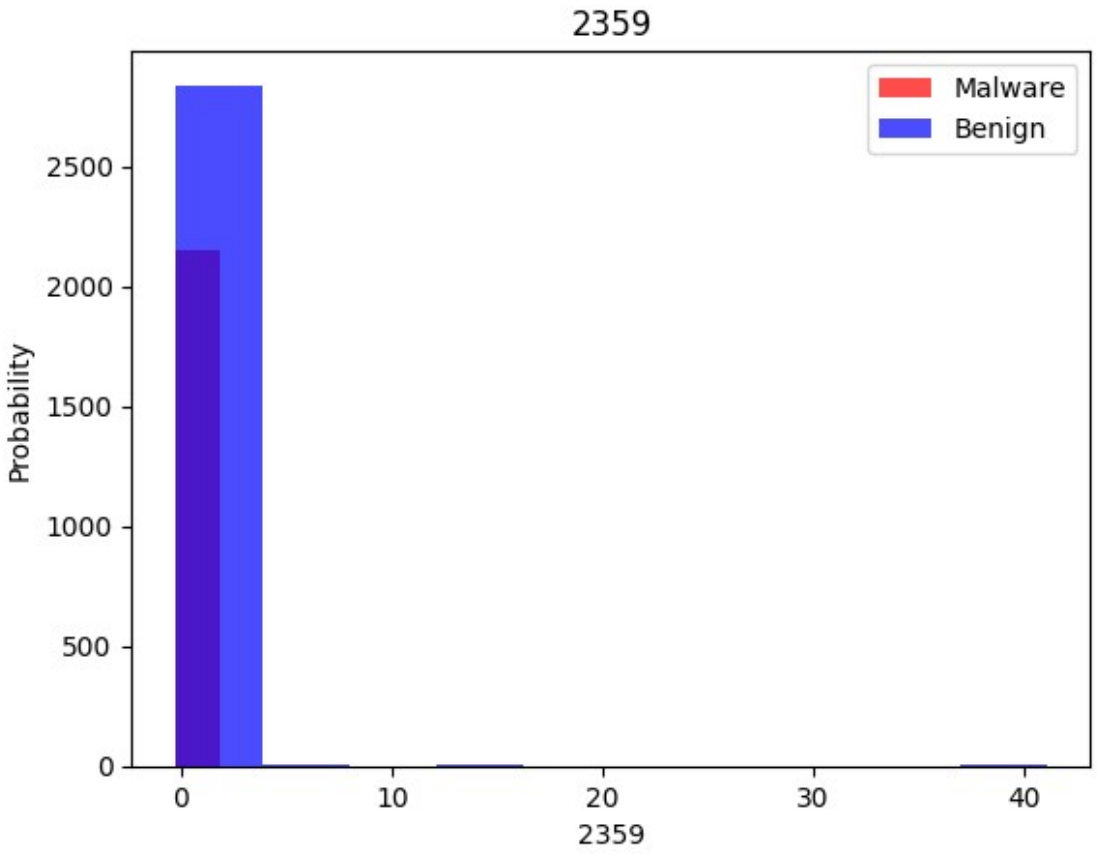
```
2     0.558562 -0.534797 -0.906729 -0.048695  ...  0.061939  0.145654

3    -1.790311  1.869867  1.102865  0.097067  ...  0.057934  0.123548

4     0.558562 -0.534797 -0.906729 -0.048695  ...  0.061939  0.145654

...        ...       ...       ...       ...  ...       ...       ...

4995  0.558562 -0.534797 -0.906729 -0.048695  ...  0.061939  0.145654

4996  0.558562 -0.534797 -0.906729 -0.048695  ...  0.061939  0.145654

4997 -1.790311 -0.534797  1.102865  0.097067  ...  0.050925  0.085101

4998 -1.790311  1.869867  1.102865  0.048480  ...  0.059936  0.136032

4999  0.558562 -0.534797 -0.906729 -0.048695  ...  0.061939  0.145654


           626      2375       640       578       691        32
681  \
0     0.069465 -0.014152  0.222049 -0.624826  0.340070 -0.146889
0.343746
1    -0.282023 -0.014143 -1.135220 -0.430832 -0.185378 -0.442819 -
0.242451
2    -0.059747 -0.014149 -1.135220 -0.569915 -0.185378 -0.347080 -
0.242451
3     0.484353 -0.014142  1.579319  0.228457 -0.185378  0.040198
0.929943
4    -1.081366 -0.014152  0.222049 -0.299002  0.340070 -0.271579 -
0.242451
...        ...       ...       ...       ...       ...       ...
...
4995 -0.101997 -0.014142 -1.135220 -0.079313 -0.185378 -0.262295 -
0.242451
4996  0.424987 -0.014152 -1.135220 -0.213667  0.340070 -0.128739 -
0.242451
4997  2.400167 -0.014110  1.579319  0.749208 -0.185378  0.128762
3.274730
4998  0.373989 -0.014147  0.222049 -0.241338 -0.185378  0.038518
0.929943
4999 -0.065376 -0.014148 -1.135220 -0.592826  0.865518  1.100478 -
0.242451


                                                      sha
0     851c1c68ce77ea256542f4b9a0b5f5240ba893e25793f0...
1     d420b98025b84150bad34dfaa23f164d8cb392b0b24ac5...
2     c24e73acbc53e0776b5937077ac50dac4fc28793be673b...
```

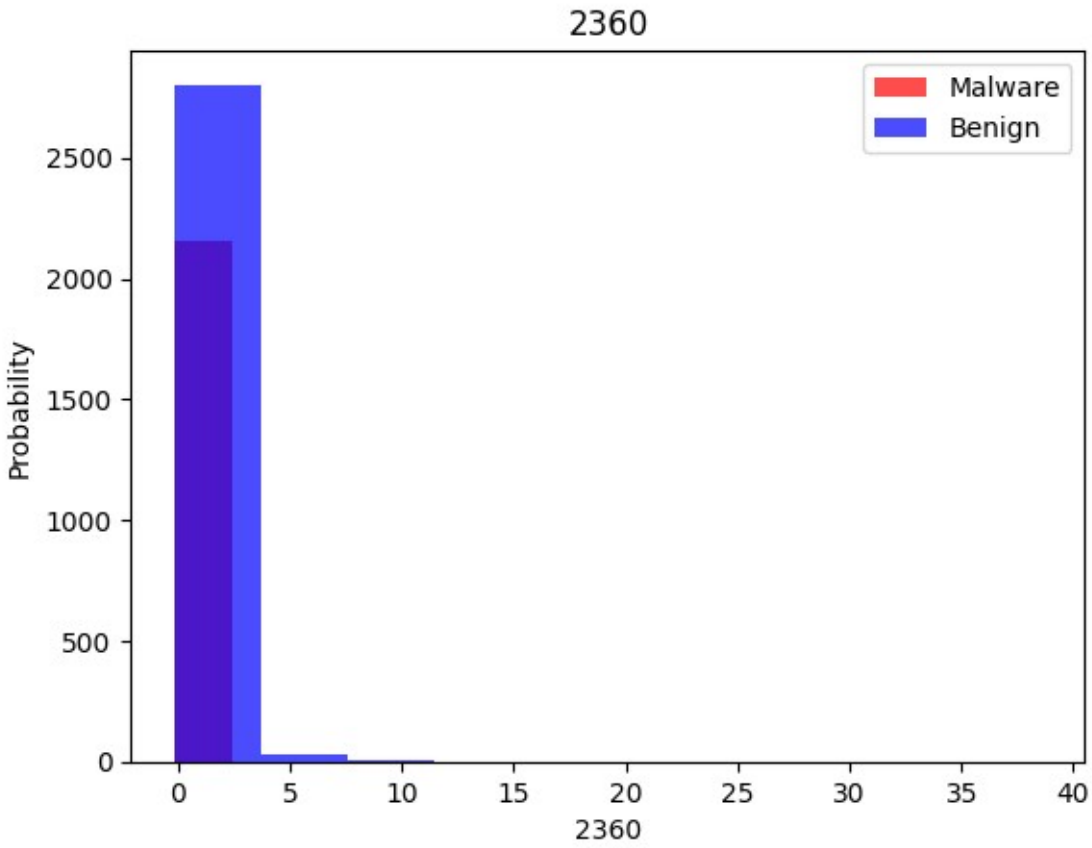
```
3     293d8b3e1e6cab75efd5c24763427f613210d4dfecdd26...
4     7551225755ec37521d0c6e15bad8ab9f806c4af7b9b7dd...
...                                                 ...
4995  9c3b4470b7426ac4c6fc23136d0af8e801cb9af0147b30...
4996  80d303cac0936a16bae673b0ab18767cf8cade686d5be4...
4997  e5c4578e36ac8fa33e5da5ee3d5dcbb98ee05b67e617f2...
4998  050074c6a7f1f577b3c9c704d385983eeb544c75096f75...
4999  e8570fdd4d817ebdf3fab40bc7b34ecc7065b579d000af...

[5000 rows x 27 columns]
```

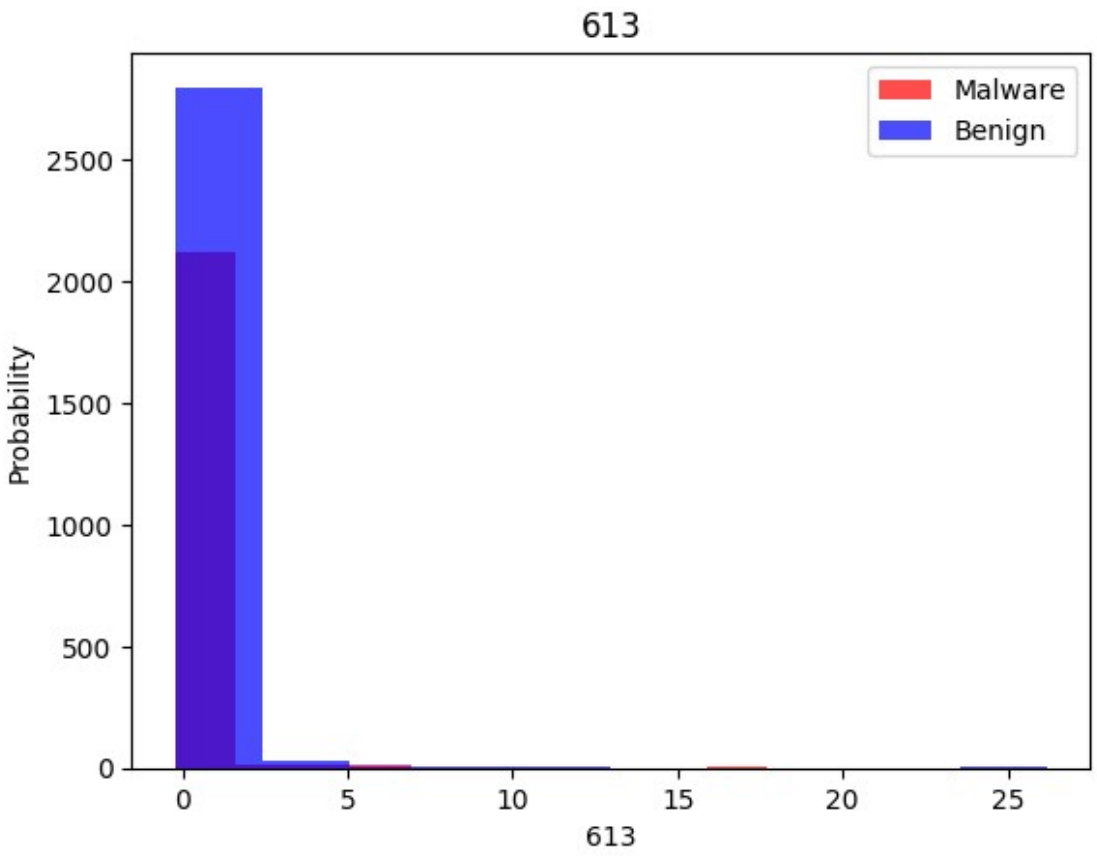
Data Analisys

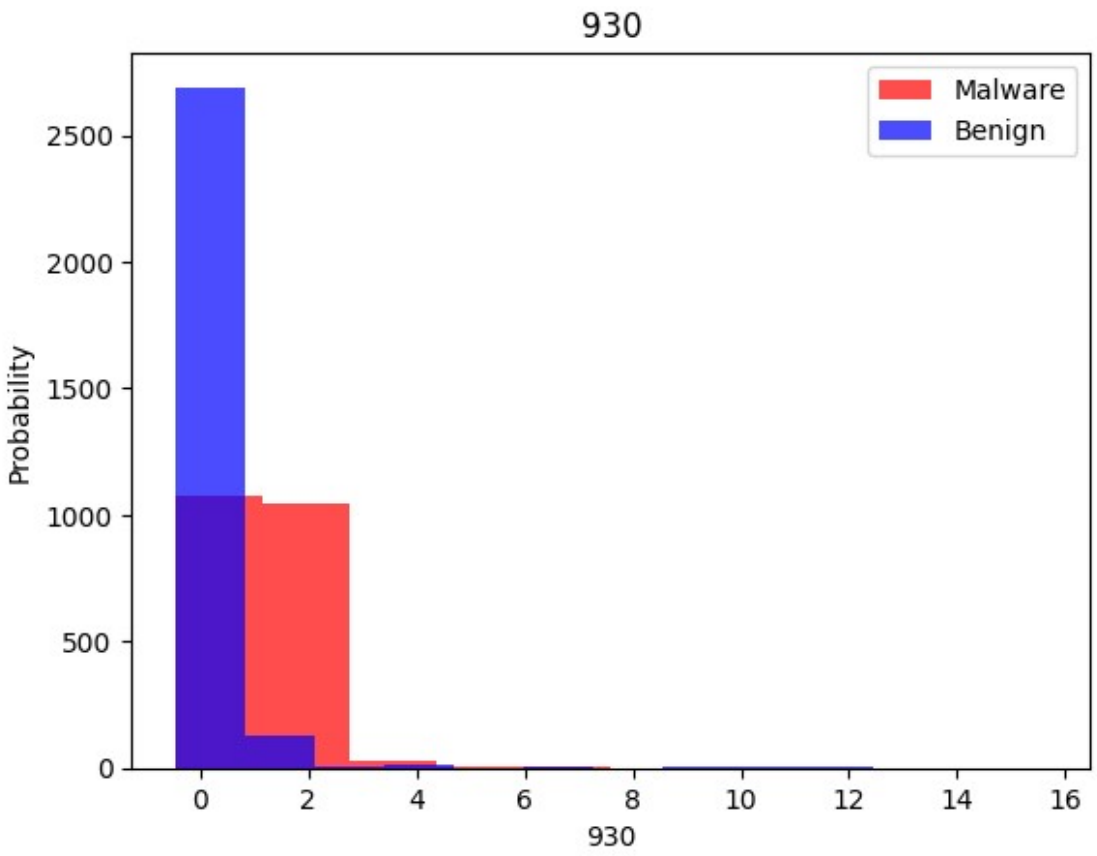
```
# viene creato un istogramma indicando le colonne blu per i software
non malevoli e le rosse per i malevoli. Itera per ogni feature e
mostra sull'asse delle ordinate la densita' delle osservazioni in base
alla feature
for feature in dataframe.columns[1:-1]:
    plt.hist(dataframe[dataframe["is_malicious"]==True]
[feature],color="red",label="Malware",alpha=0.7)
    plt.hist(dataframe[dataframe["is_malicious"]==False]
[feature],color="blue",label="Benign",alpha=0.7)
    plt.title(feature)
    plt.ylabel("Density")
    plt.xlabel(feature)
    plt.legend()
    plt.show()
```

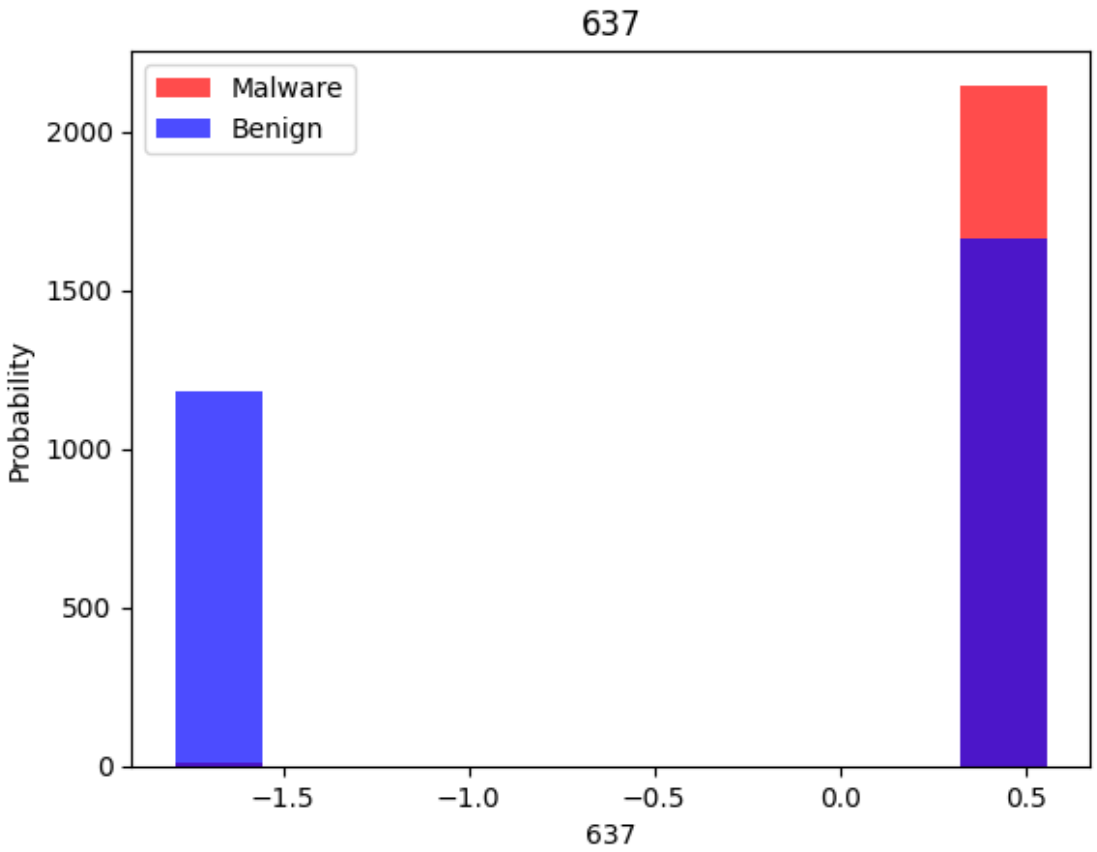

658
Malware
Benign
1750
1500
1250
1000
750
500
250
0
Probability
−2
−1
0
1
2
658

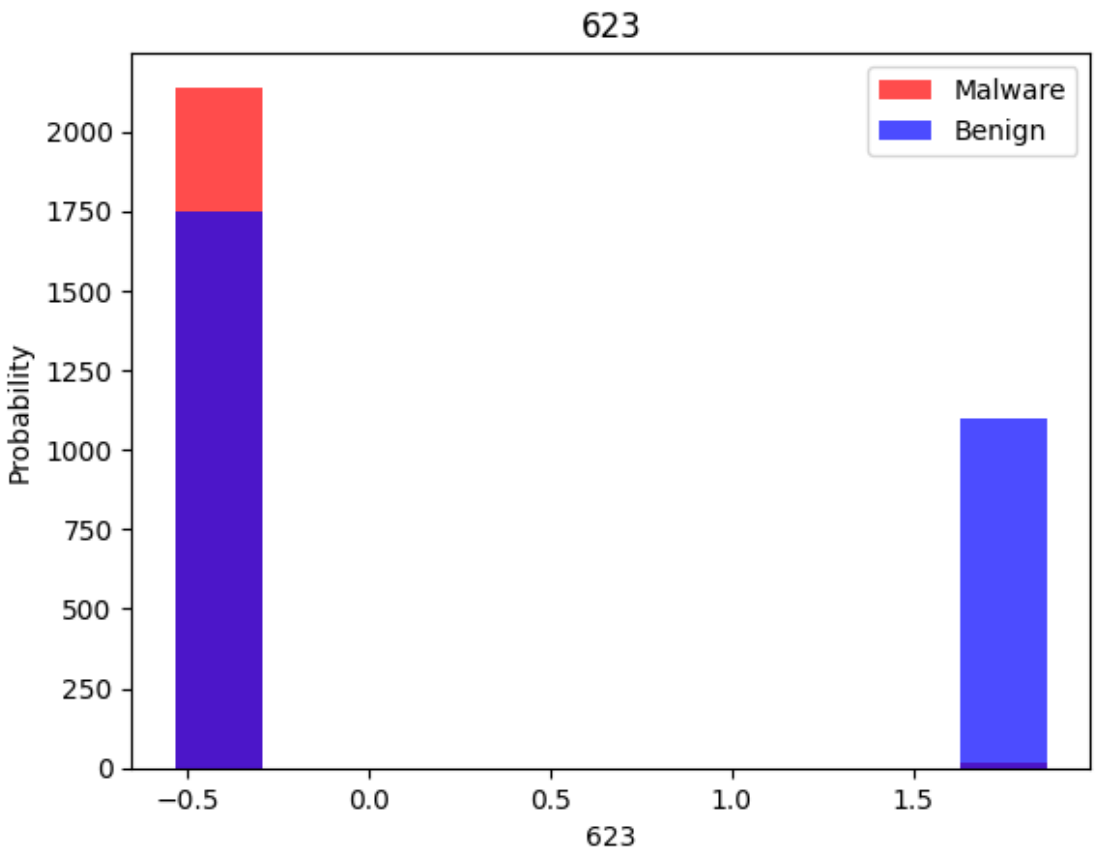

2359
Malware
Benign
Probability
2500
2000
1500
1000
500
0
0
10
20
30
40
2359

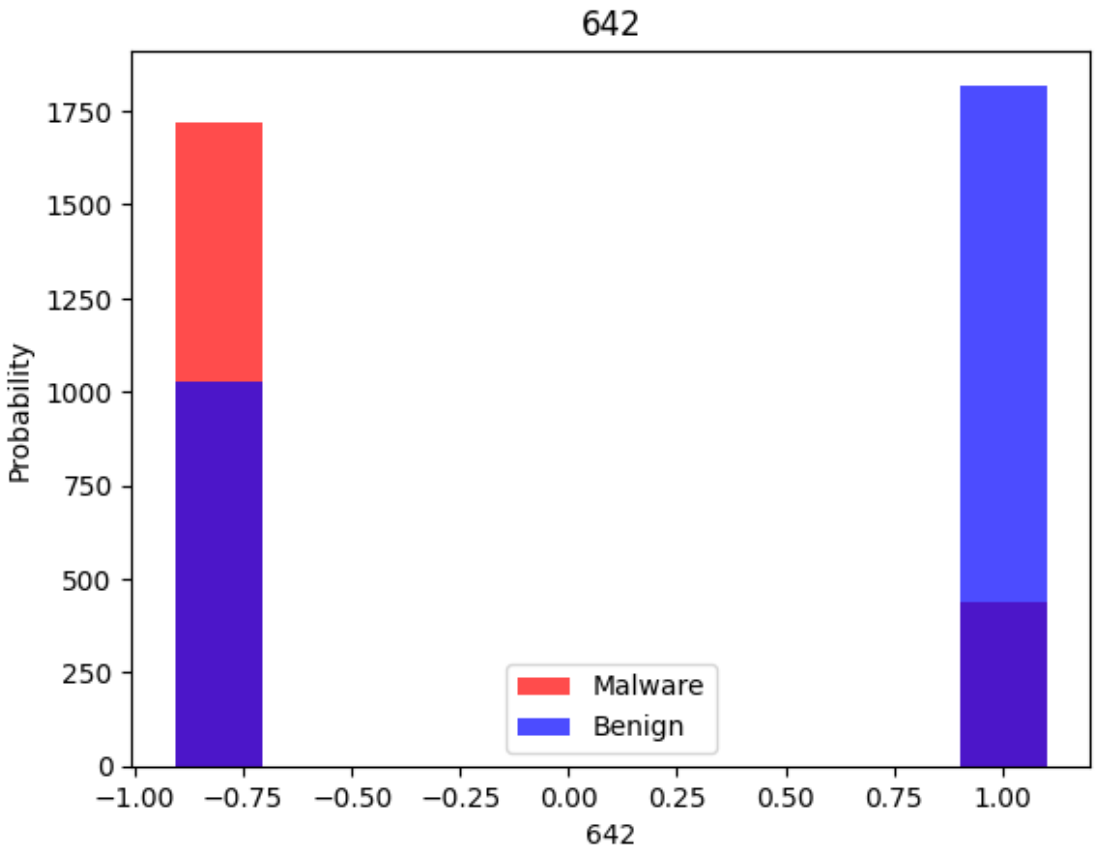

2360
Malware
Benign
Probability
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
35
40
2360

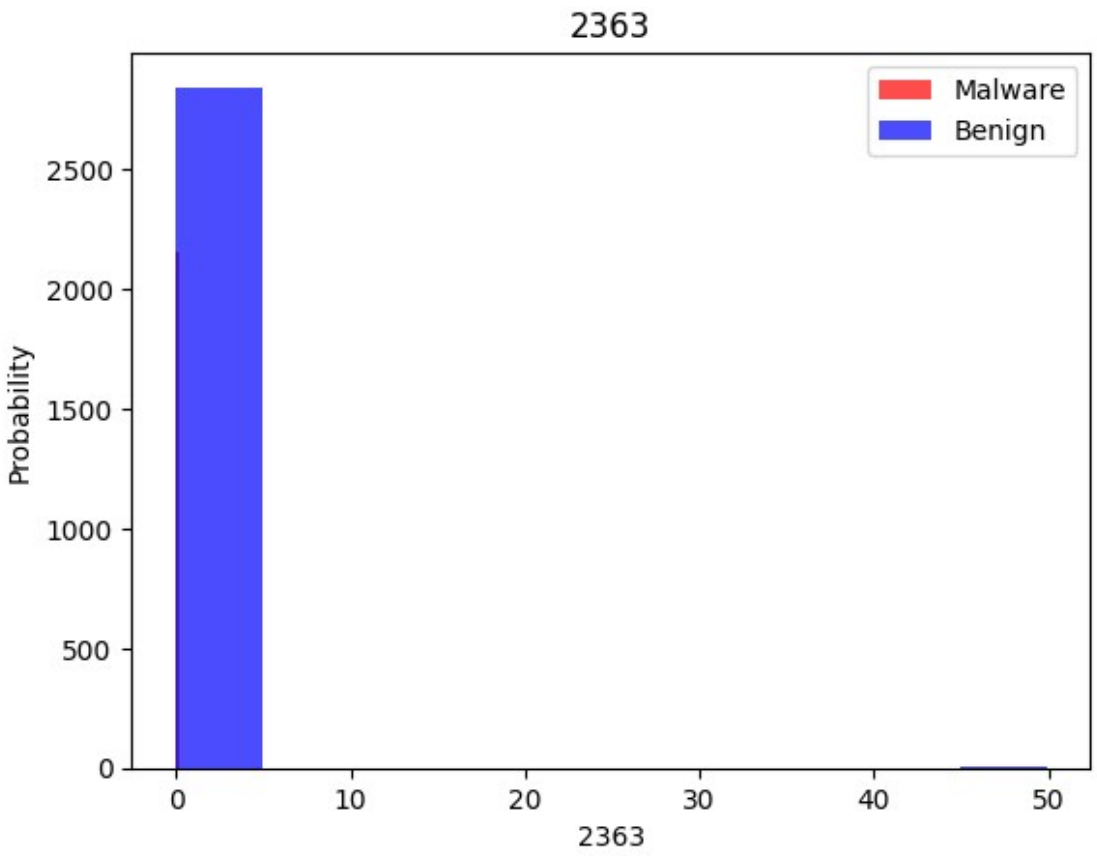

613
Malware
Benign
2500
2000
1500
1000
500
0
Probability
0
5
10
15
20
25
613

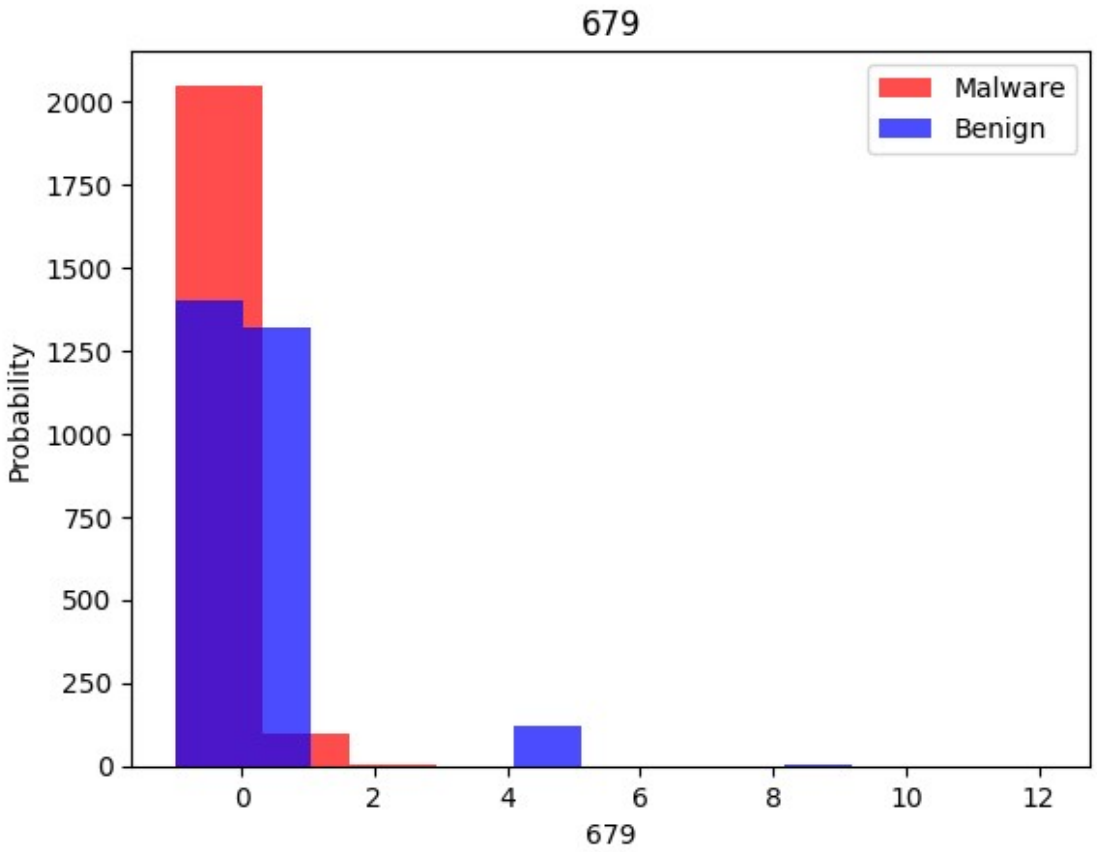

930
Malware
Benign
Probability
2500
2000
1500
1000
500
0
0
2
4
6
8
10
12
14
16
930

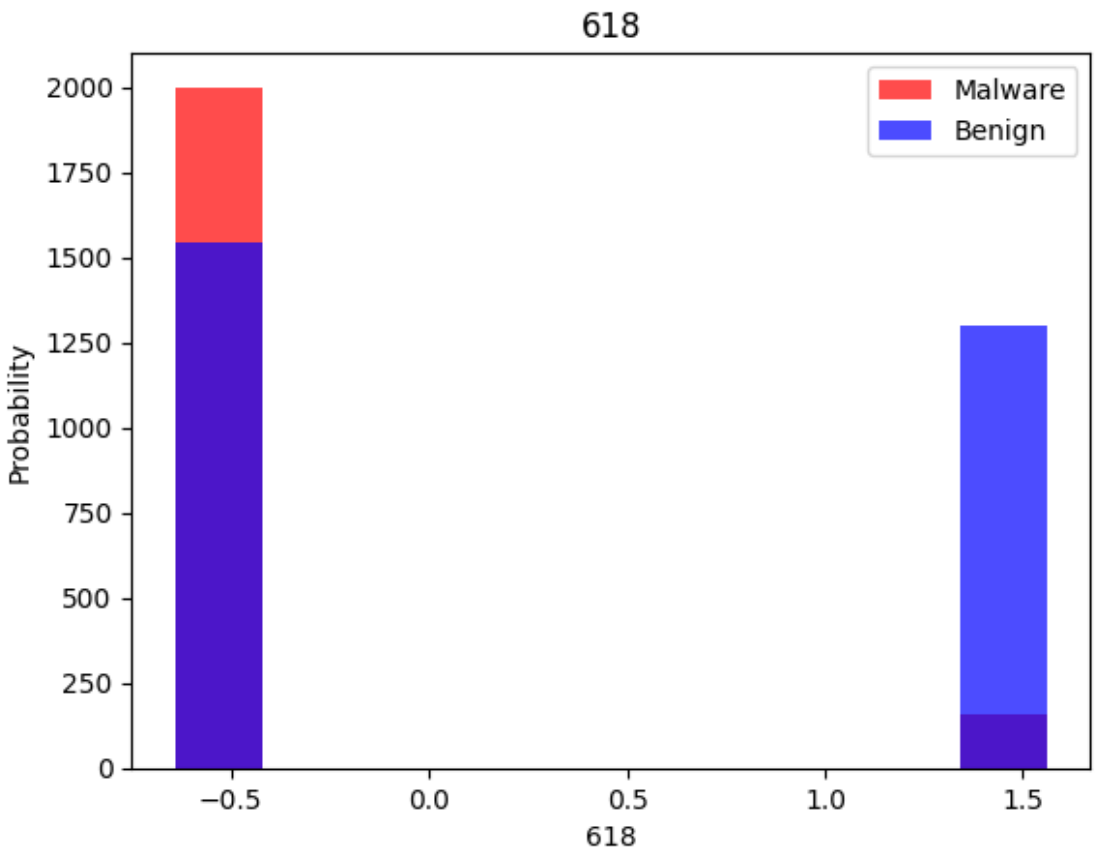

637
Malware
Benign
2000
1500
1000
500
0
Probability
−1.5
−1.0
−0.5
0.0
0.5
637

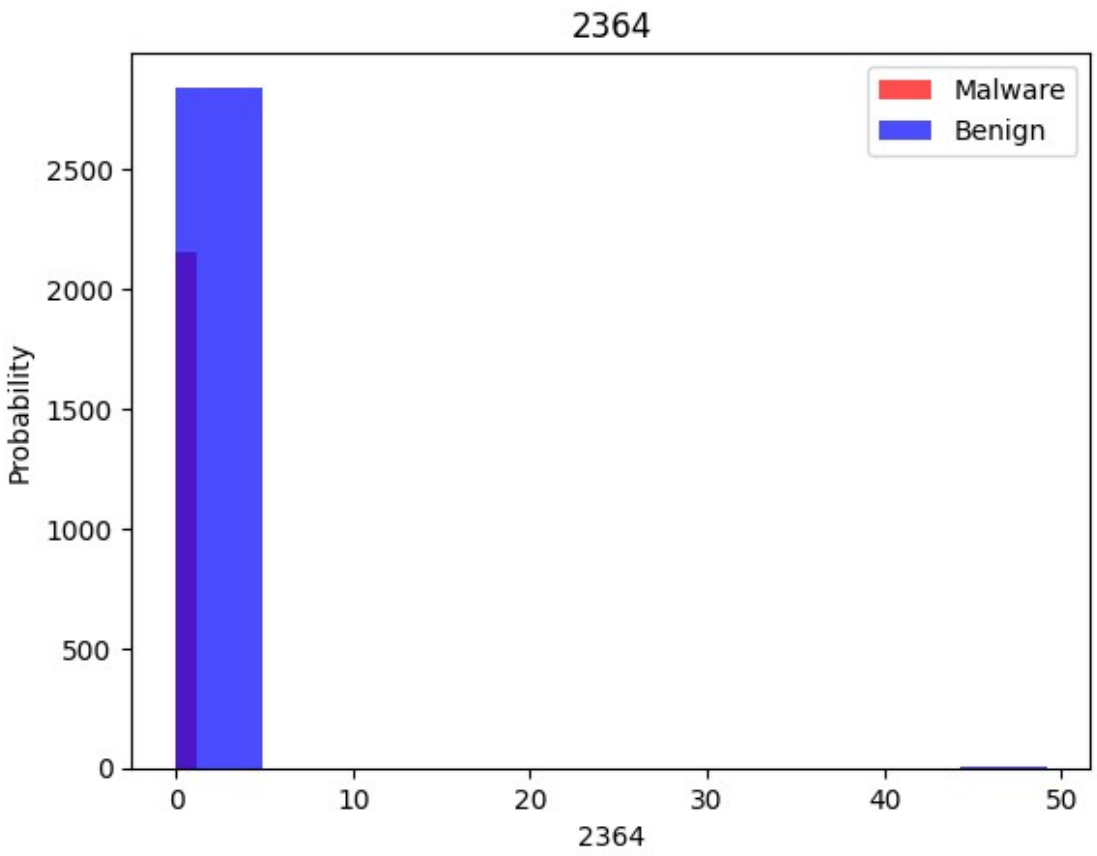

623
Malware
Benign
Probability
2000
1750
1500
1250
1000
750
500
250
0
−0.5
0.0
0.5
1.0
1.5
623

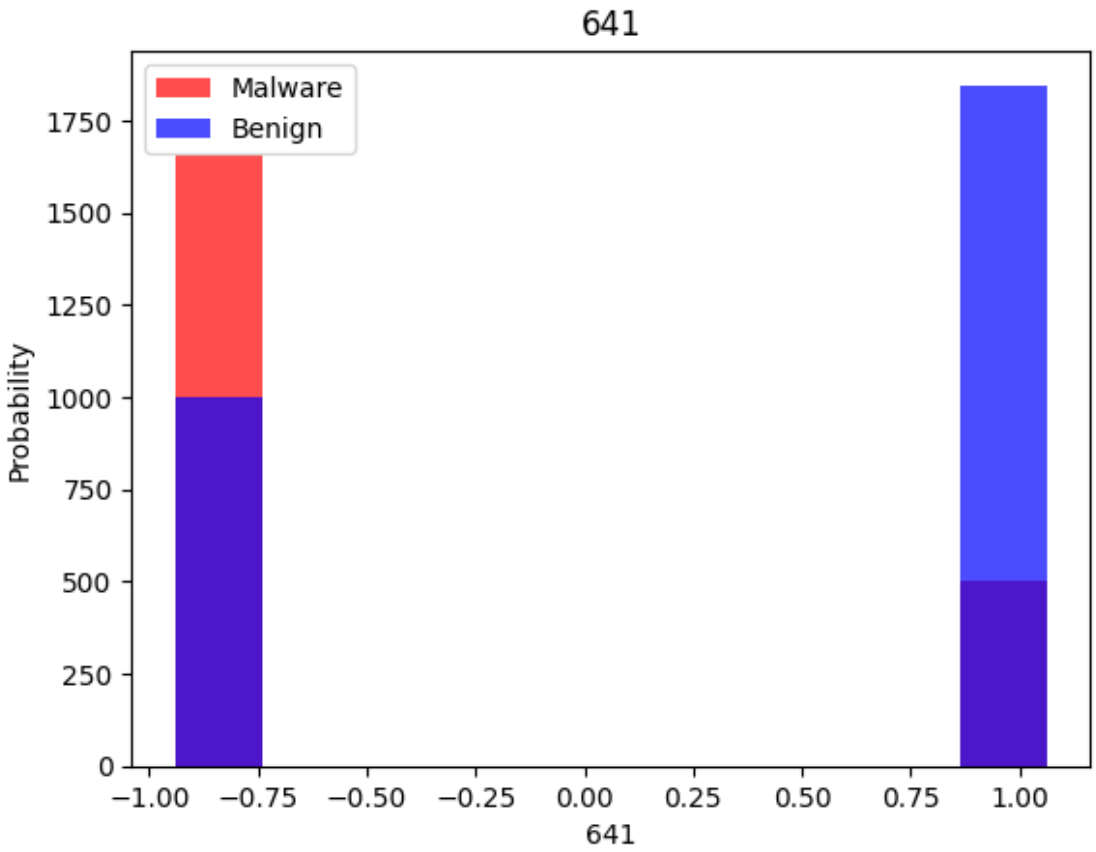

642
Probability
1750
1500
1250
1000
750
500
250
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
642
Malware
Benign

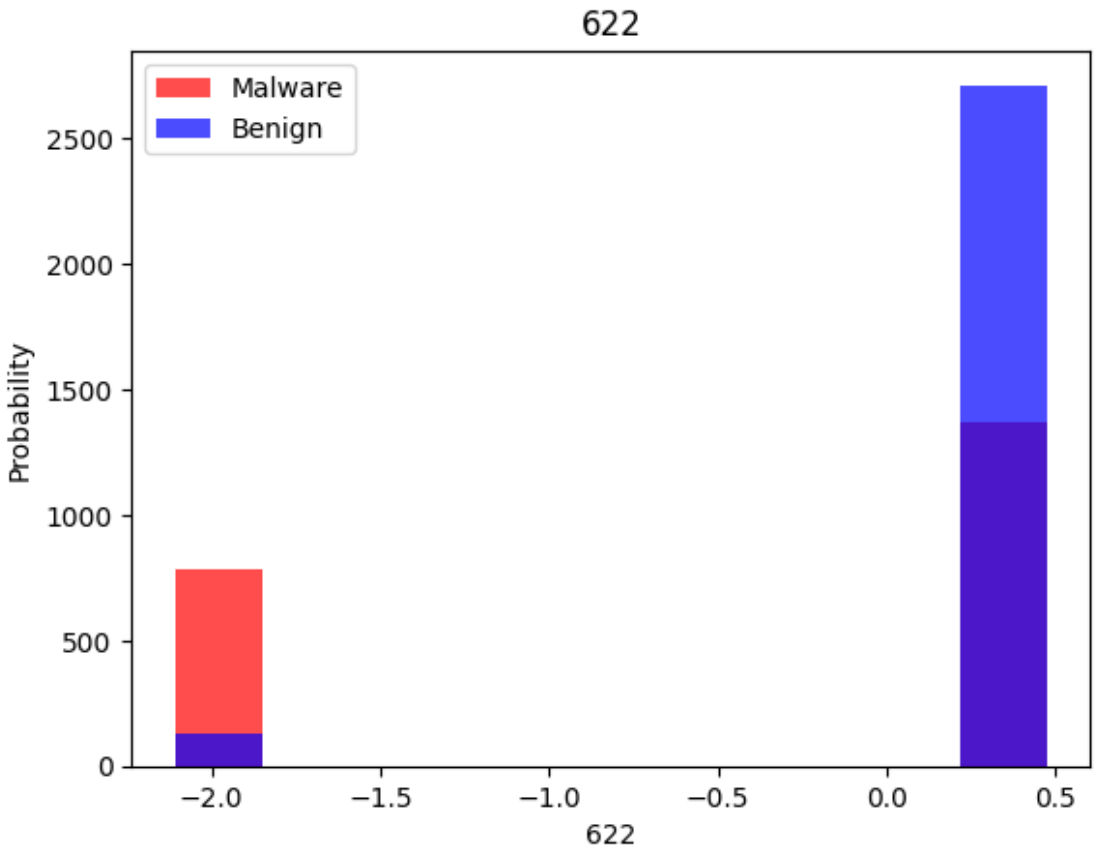

2363
Malware
Benign
Probability
2500
2000
1500
1000
500
0
0
10
20
30
40
50
2363

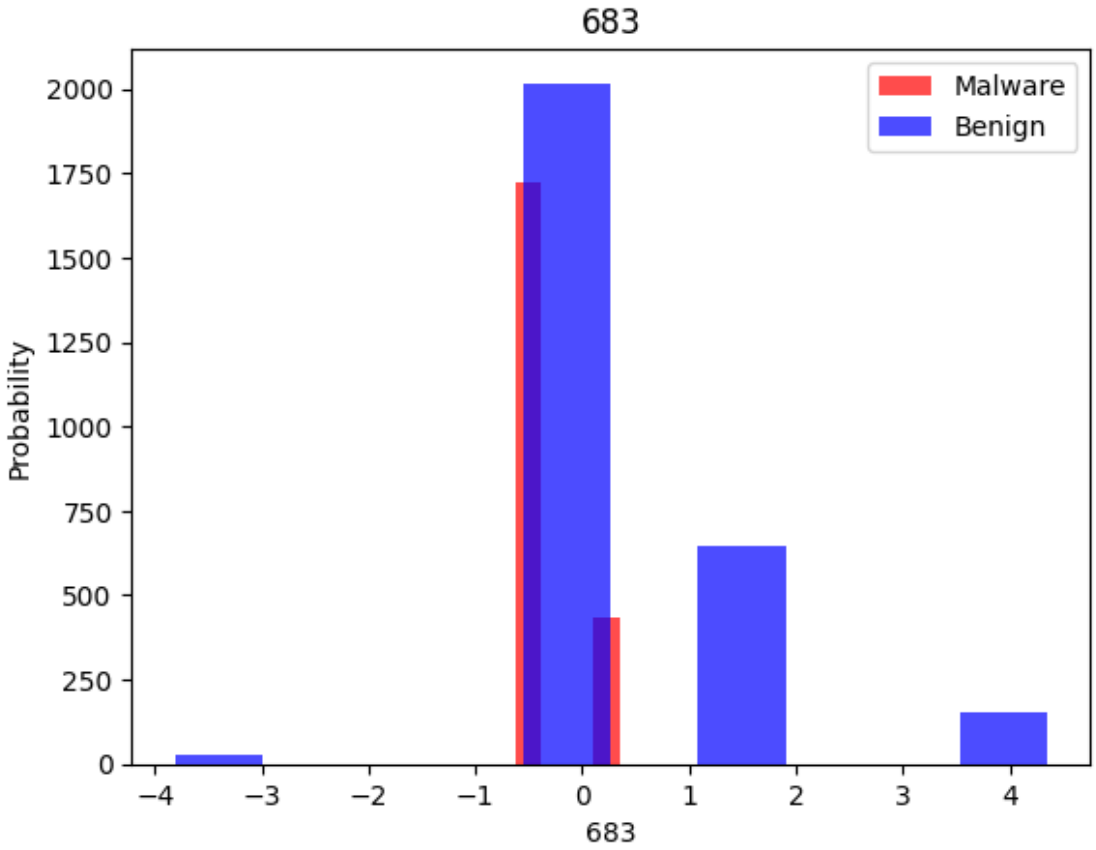

679
Malware
Benign
Probability
2000
1750
1500
1250
1000
750
500
250
0
0
2
4
6
8
10
12
679

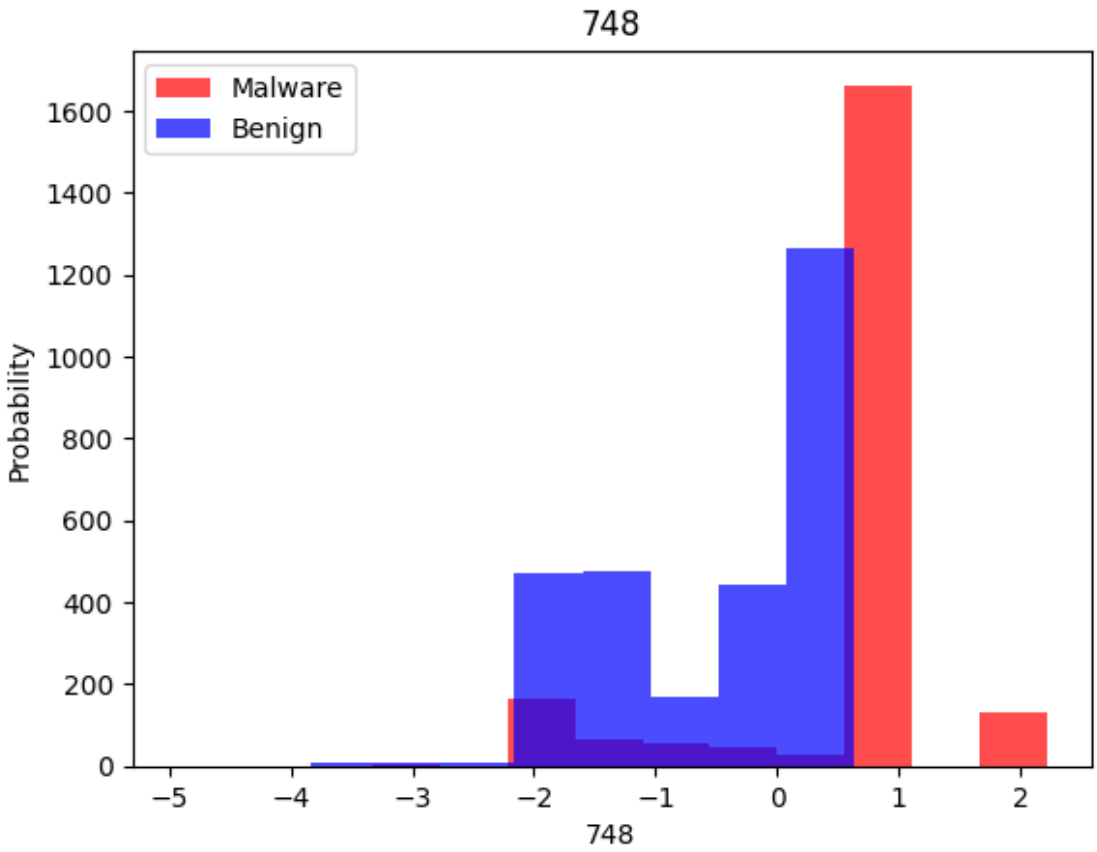

618
Malware
Benign
Probability
2000
1750
1500
1250
1000
750
500
250
0
−0.5
0.0
0.5
1.0
1.5
618

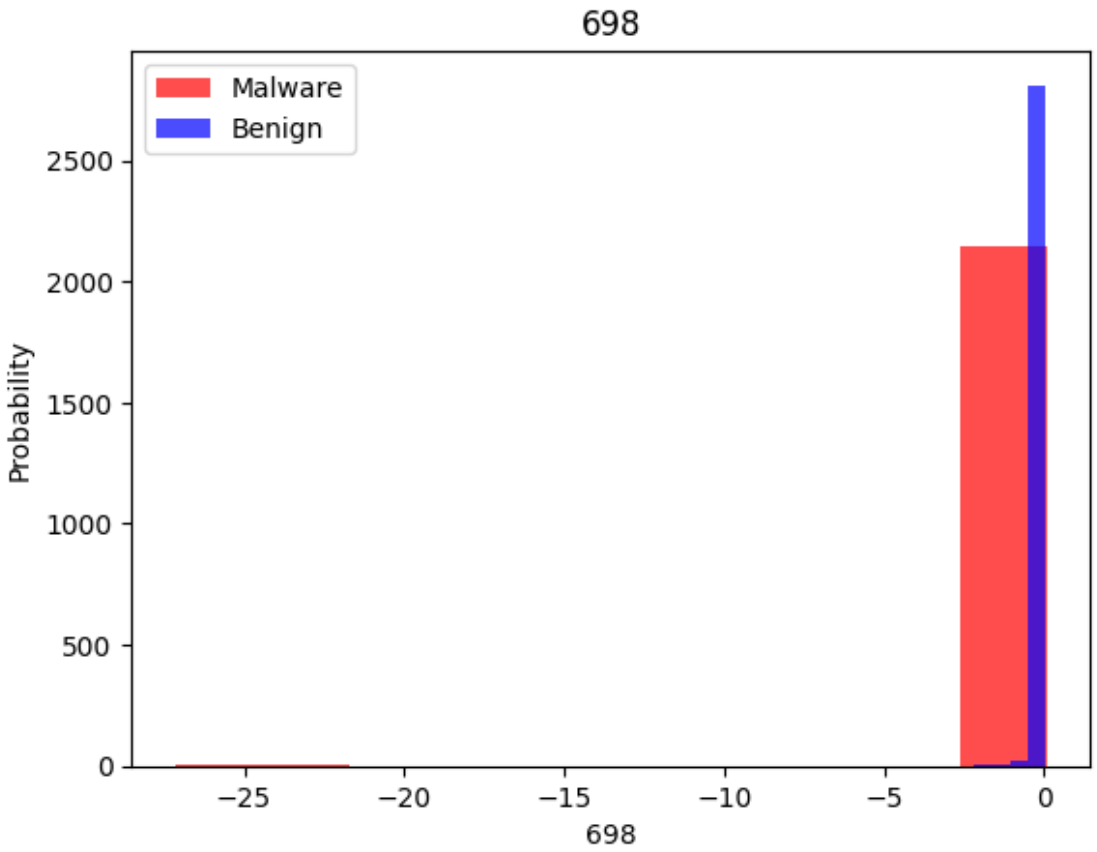

2364
Malware
Benign
Probability
2500
2000
1500
1000
500
0
0
10
20
30
40
50
2364

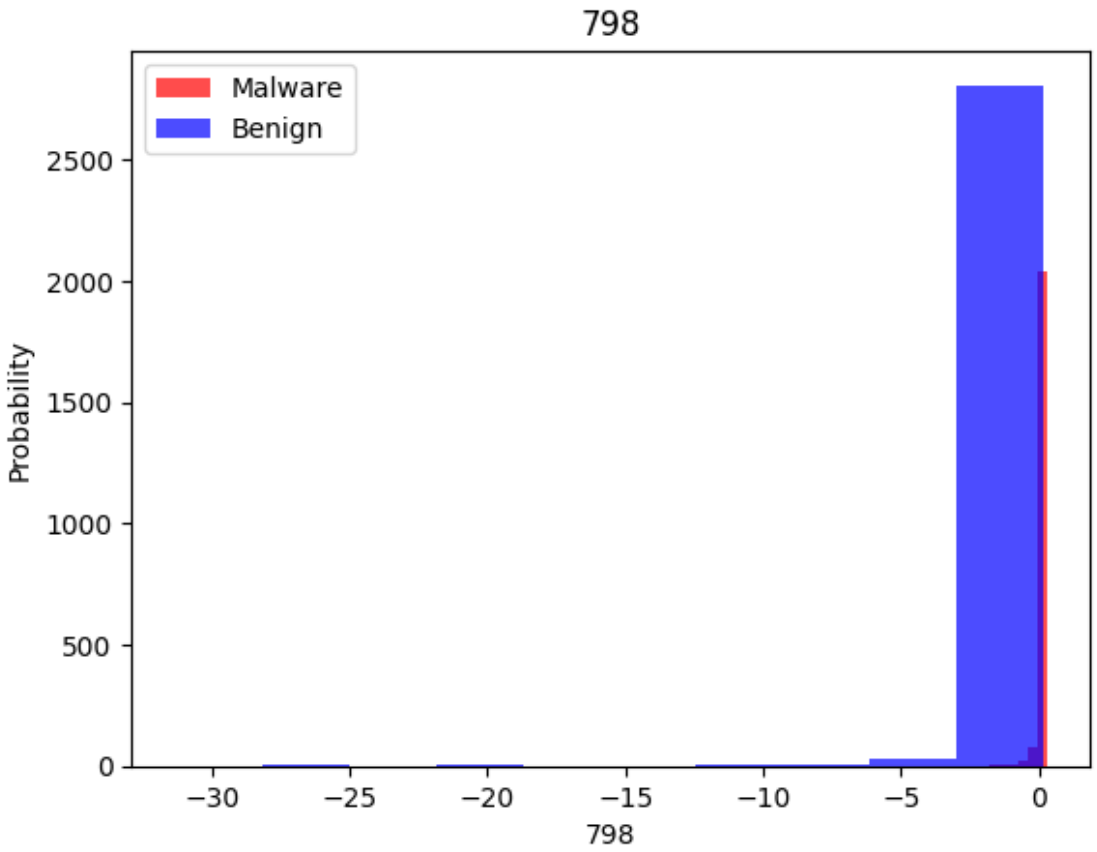

641
Malware
Benign
1750
1500
1250
1000
750
500
250
0
Probability
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
641

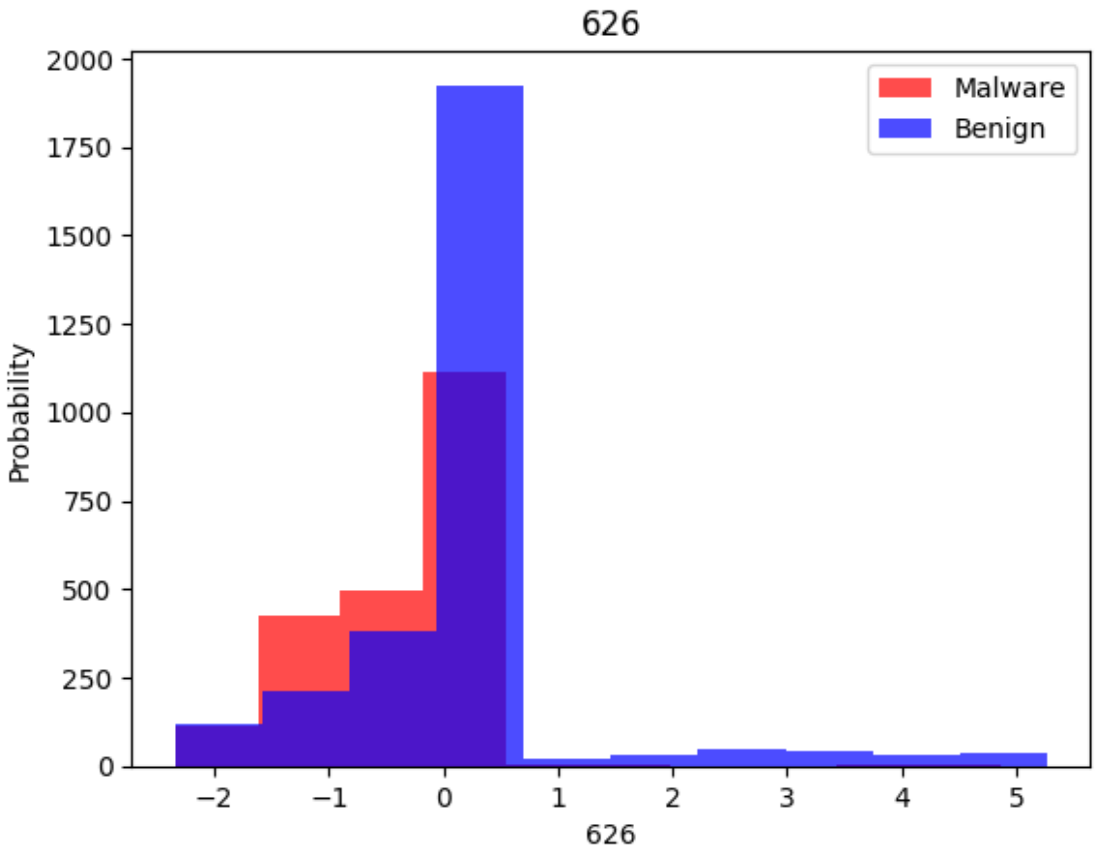

622
Malware
Benign
2500
2000
1500
1000
500
0
Probability
−2.0
−1.5
−1.0
−0.5
0.0
0.5
622

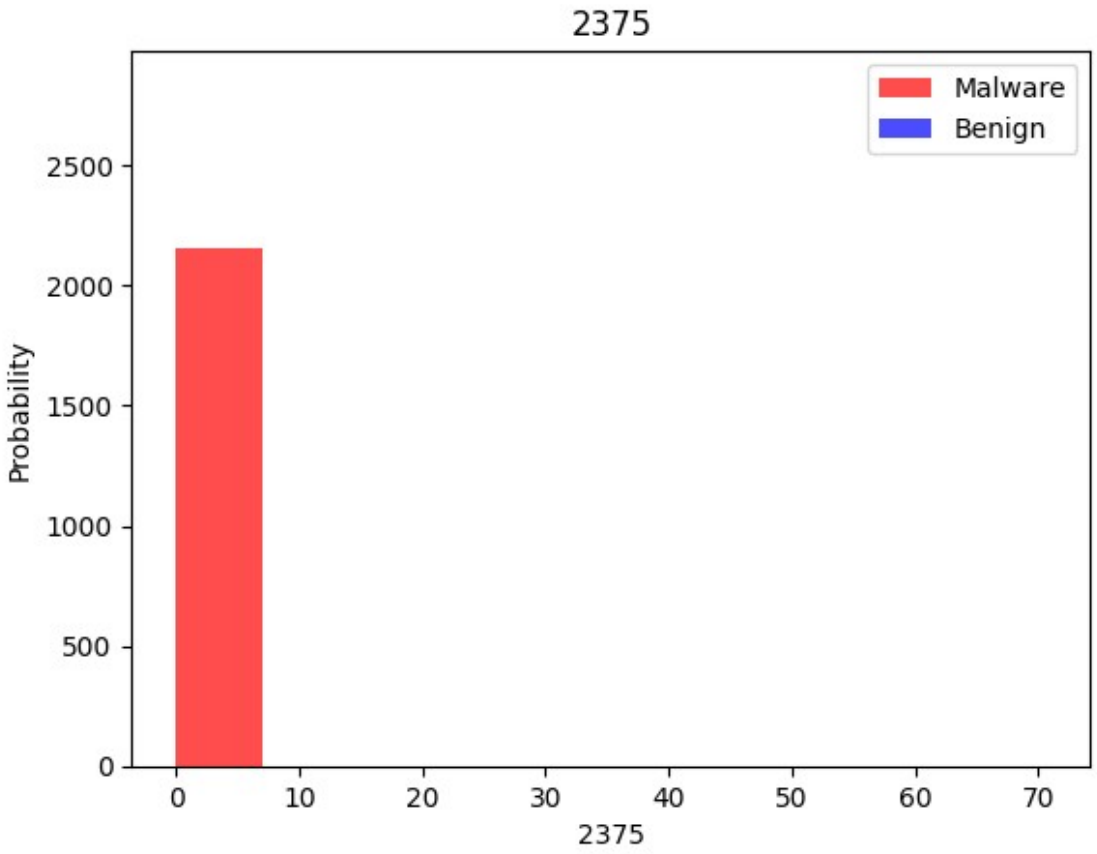

683
Malware
Benign
Probability
2000
1750
1500
1250
1000
750
500
250
0
−4
−3
−2
−1
0
1
2
3
4
683

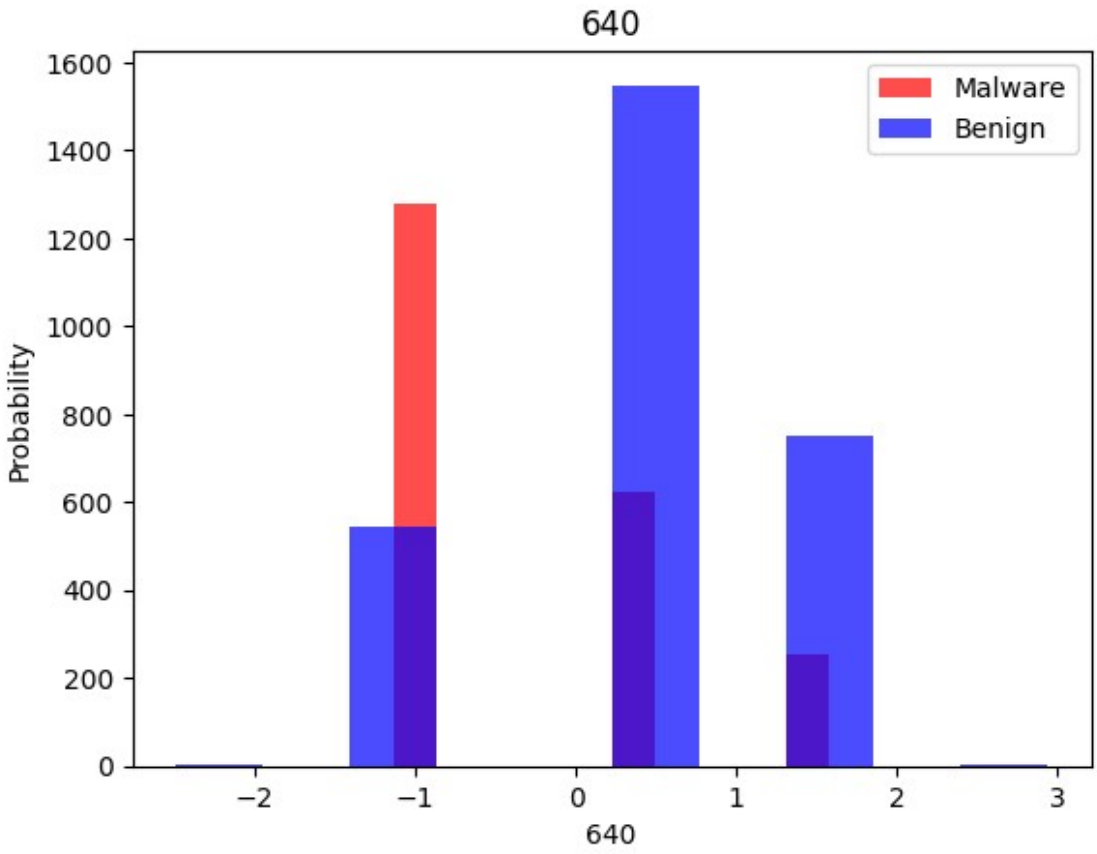

748
Malware
Benign
Probability
1600
1400
1200
1000
800
600
400
200
0
−5
−4
−3
−2
−1
0
1
2
748

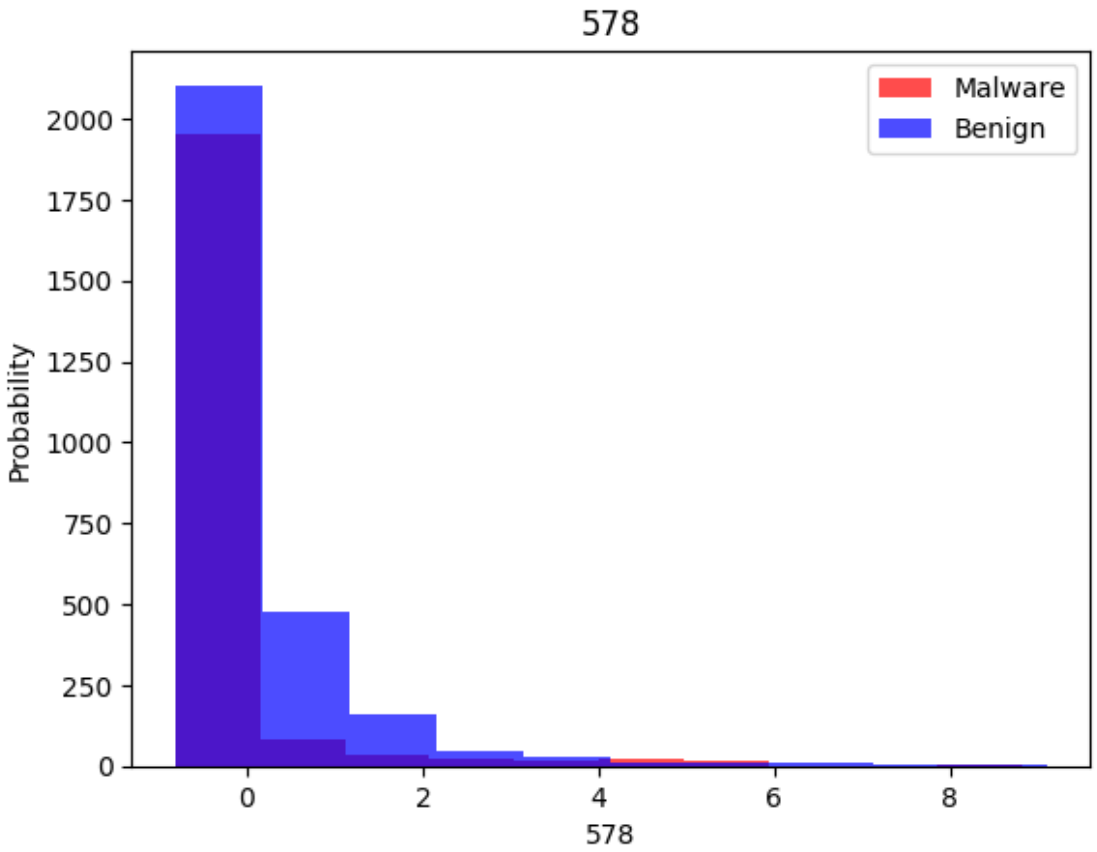

698
Malware
Benign
2500
2000
1500
1000
500
0
Probability
−25
−20
−15
−10
−5
0
698

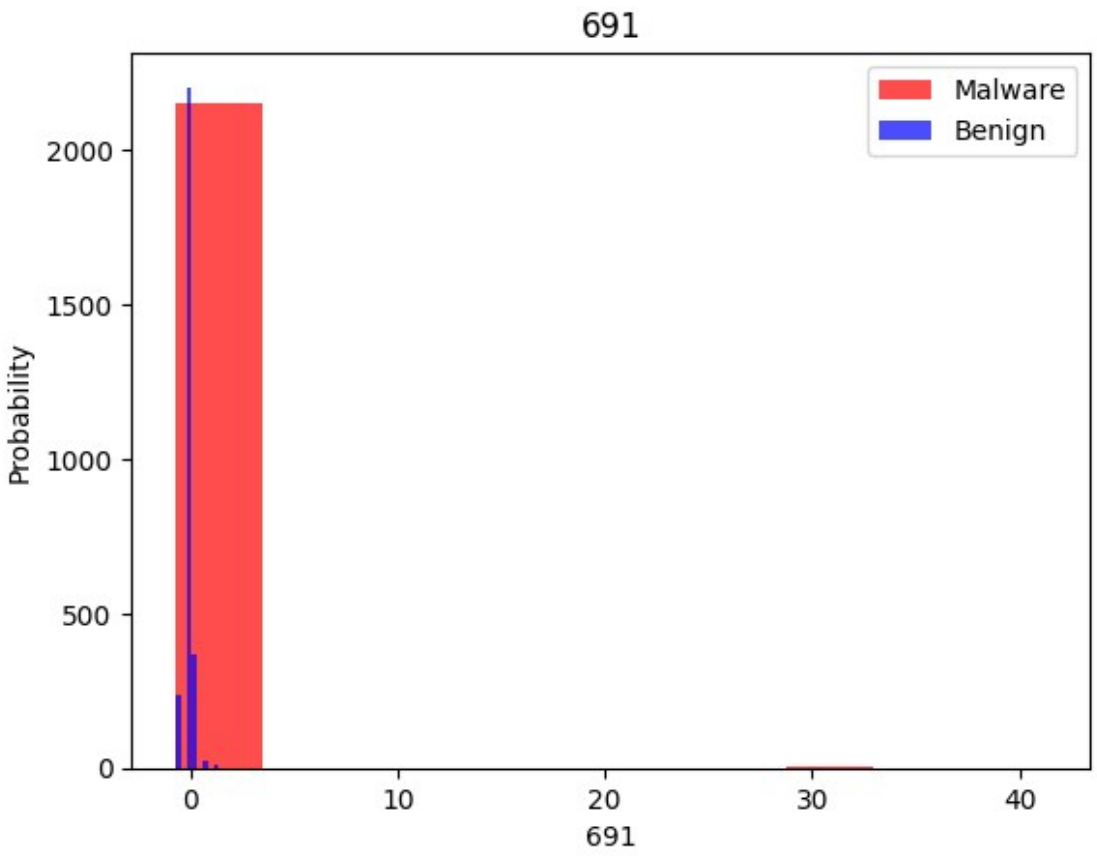

798
Malware
Benign
2500
2000
1500
1000
500
0
Probability
−30
−25
−20
−15
−10
−5
0
798

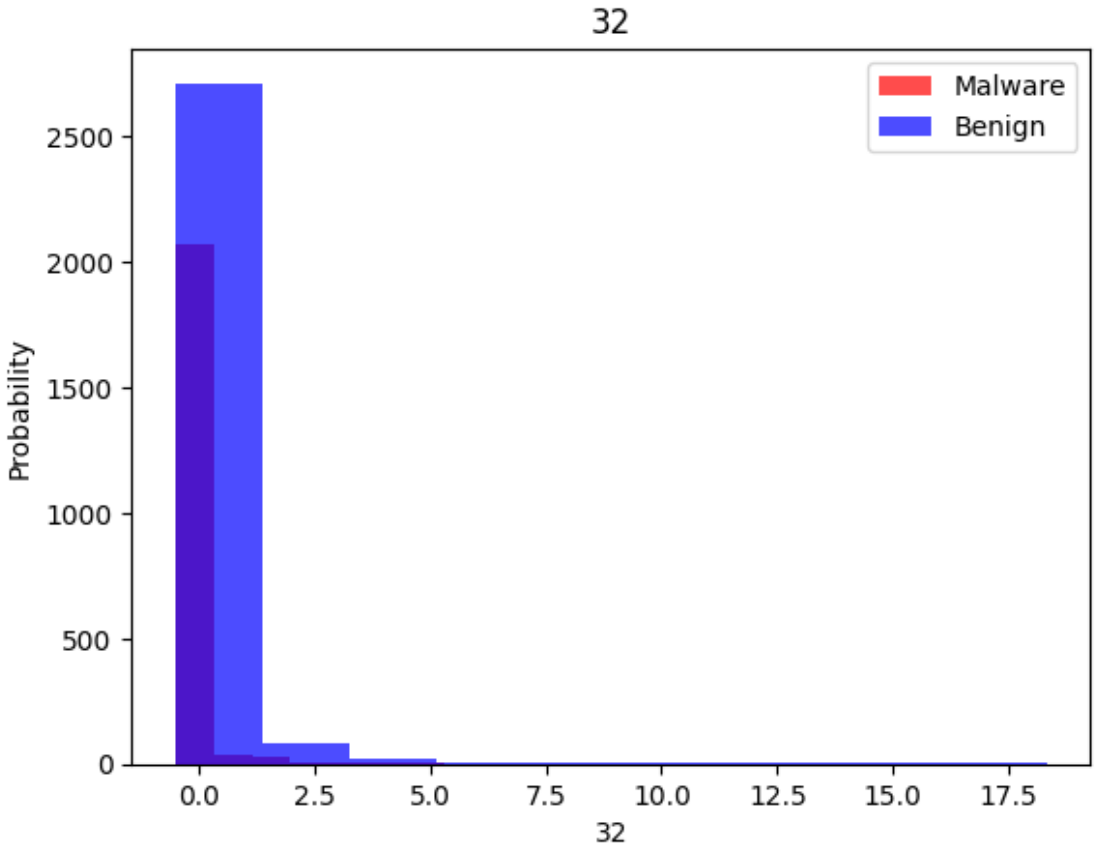

626
Malware
Benign
Probability
2000
1750
1500
1250
1000
750
500
250
0
−2
−1
0
1
2
3
4
5
626

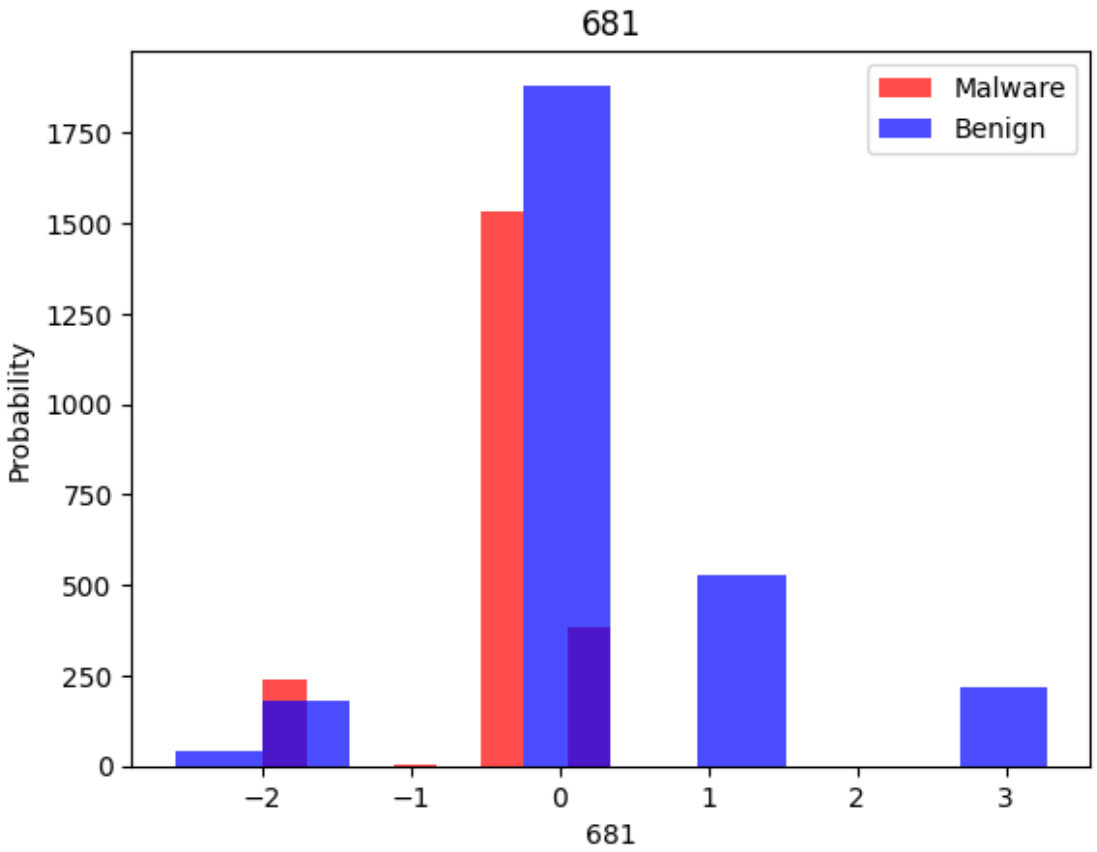

2375
Malware
Benign
2500
2000
1500
1000
500
0
Probability
0
10
20
30
40
50
60
70
2375


640
Malware
Benign
Probability
1600
1400
1200
1000
800
600
400
200
0
−2
−1
0
1
2
3
640


578
Malware
Benign
2000
1750
1500
1250
1000
750
500
250
0
Probability
0
2
4
6
8
578


691
Malware
Benign
2000
1500
1000
500
0
Probability
0
10
20
30
40
691


32
Malware
Benign
Probability
2500
2000
1500
1000
500
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
32



Dataset setup

```
#splitta il dataset in training e test dividendo le features e la
target
X = dataframe.iloc[:,1:-1]
y = dataframe.iloc[:,:1]
X = X.values
y = y.values
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

Decision Trees

```
#istanzia e addestra un albero di decisione
from sklearn.tree import DecisionTreeClassifier

decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train,np.ravel(y_train))
y_predicted = decision_tree_model.predict(X_test)

#Performance
print(classification_report(y_predicted,y_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       546
           1       0.96      0.94      0.95       454

    accuracy                           0.95      1000
   macro avg       0.95      0.95      0.95      1000
weighted avg       0.95      0.95      0.95      1000
```

Logistic Regression

```
#istanzia e addestra un modello di regressione logistica
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression()
logistic_model.fit(X_train,np.ravel(y_train))
y_predicted = logistic_model.predict(X_test)


#Performance
print(classification_report(y_predicted,y_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       540
           1       0.90      0.87      0.88       460

    accuracy                           0.89      1000
   macro avg       0.89      0.89      0.89      1000
weighted avg       0.89      0.89      0.89      1000
```

Support Vector Classifier

```
#istanzia e addestra un modello di SVM
from sklearn.svm import SVC

svc_model = SVC(C=3)
svc_model.fit(X_train,np.ravel(y_train))
y_predicted = svc_model.predict(X_test)

#Performance
print(classification_report(y_predicted,y_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       548
           1       0.95      0.94      0.94       452

    accuracy                           0.95      1000
```

```
   macro avg       0.95      0.95      0.95      1000
weighted avg       0.95      0.95      0.95      1000
```

Random Forest

```
#istanzia e addestra un modello di Random Forest
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()
rf_model.fit(X_train,np.ravel(y_train))
y_predicted = rf_model.predict(X_test)

#Performance
print(classification_report(y_predicted,y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       559
           1       0.96      0.97      0.96       441

    accuracy                           0.97      1000
   macro avg       0.97      0.97      0.97      1000
weighted avg       0.97      0.97      0.97      1000
```

Stacking

```
# istanzia un classificatore a stack prendendo come input i modelli
precedentemente allentati. NB: il metamodello scelto e di default e'
regressione logistica
from sklearn.ensemble import StackingClassifier

estimators = [
    ("decision tees",decision_tree_model),
    ("logistic Regression",logistic_model),
    ("random forest",rf_model),
    ("svc",svc_model),

]

stacking_model = StackingClassifier(estimators=estimators)

stacking_model.fit(X_train,np.ravel(y_train))
y_predicted = stacking_model.predict(X_test)
print(classification_report(y_predicted, y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       562
           1       0.96      0.97      0.96       438

    accuracy                           0.97      1000
   macro avg       0.97      0.97      0.97      1000
weighted avg       0.97      0.97      0.97      1000
```

K-fold Validation

```
#accuratezza del classificatore Stack sul training set con tecnica
kfold e k = 10
score = cross_val_score(stacking_model,X_train,np.ravel(y_train),cv =
10,scoring="accuracy")
print(np.mean(score))
```

```
0.9712500000000001
```

```
#accuratezza del classificatore Stack sul test set con tecnica kfold e
k = 10

test_score = cross_val_score(stacking_model,X_test,np.ravel(y_test),cv
= 10,scoring="accuracy")
print(np.mean(test_score))
```

```
0.954
```

Exporting Classifier

```
#esportazione del modello addestrato
import pickle

with open("../../Model/classifier.pkl","wb") as f:
    pickle.dump(stacking_model,f)
```