mini1

May 11, 2024

[2]:
```
import os
import xgboost as xgb
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, TensorDataset
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier

from tensorflow.keras import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential



from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, f1_score, precision_score,␣
 ↪recall_score



from tqdm import tqdm
data = pd.read_csv(r'C:\Users\BRINDHA\Desktop\web-page-phishing.csv')
data
```

[2]:
```
        url_length  n_dots  n_hypens  n_underline  n_slash  n_questionmark  \
0               37       3         0            0        0               0
```

```
1               77       1         0            0        0               0
2              126       4         1            2        0               1
3               18       2         0            0        0               0
4               55       2         2            0        0               0
…              …     …       …          …       …              …
100072          23       3         1            0        0               0
100073          34       2         0            0        0               0
100074          70       2         1            0        5               0
100075          28       2         0            0        1               0
100076          16       2         0            0        0               0

        n_equal  n_at  n_and  n_exclamation  n_space  n_tilde  n_comma  \
0             0     0      0              0        0        0        0
1             0     0      0              0        0        0        0
2             3     0      2              0        0        0        0
3             0     0      0              0        0        0        0
4             0     0      0              0        0        0        0
…           …   …     …              …       …       …       …
100072        0     0      0              0        0        0        0
100073        0     0      0              0        0        0        0
100074        0     0      0              0        0        0        0
100075        0     0      0              0        0        0        0
100076        0     0      0              0        0        0        0

        n_plus  n_asterisk  n_hastag  n_dollar  n_percent  n_redirection  \
0            0           0         0         0          0              0
1            0           0         0         0          0              1
2            0           0         0         0          0              1
3            0           0         0         0          0              1
4            0           0         0         0          0              1
…          …         …       …       …         …              …
100072       0           0         0         0          0              0
100073       0           0         0         0          0              2
100074       0           0         0         0          0              0
100075       0           0         0         0          0              0
100076       0           0         0         0          0              0

        phishing
0              0
1              1
2              1
3              0
4              0
…            …
100072         0
100073         0
100074         1
```

```
100075          1
100076          0

[100077 rows x 20 columns]
```

[3]:
```
for col in data.columns:
  print(f'unique({col}) = {len(data[col].unique())}')
```

```
unique(url_length) = 490
unique(n_dots) = 23
unique(n_hypens) = 33
unique(n_underline) = 22
unique(n_slash) = 25
unique(n_questionmark) = 6
unique(n_equal) = 22
unique(n_at) = 15
unique(n_and) = 21
unique(n_exclamation) = 10
unique(n_space) = 10
unique(n_tilde) = 6
unique(n_comma) = 8
unique(n_plus) = 10
unique(n_asterisk) = 18
unique(n_hastag) = 7
unique(n_dollar) = 10
unique(n_percent) = 53
unique(n_redirection) = 15
unique(phishing) = 2
```

[3]:
```
is_there_null_values =  data.isna().any().any()
is_there_null_values
```

[3]:
```
False
```

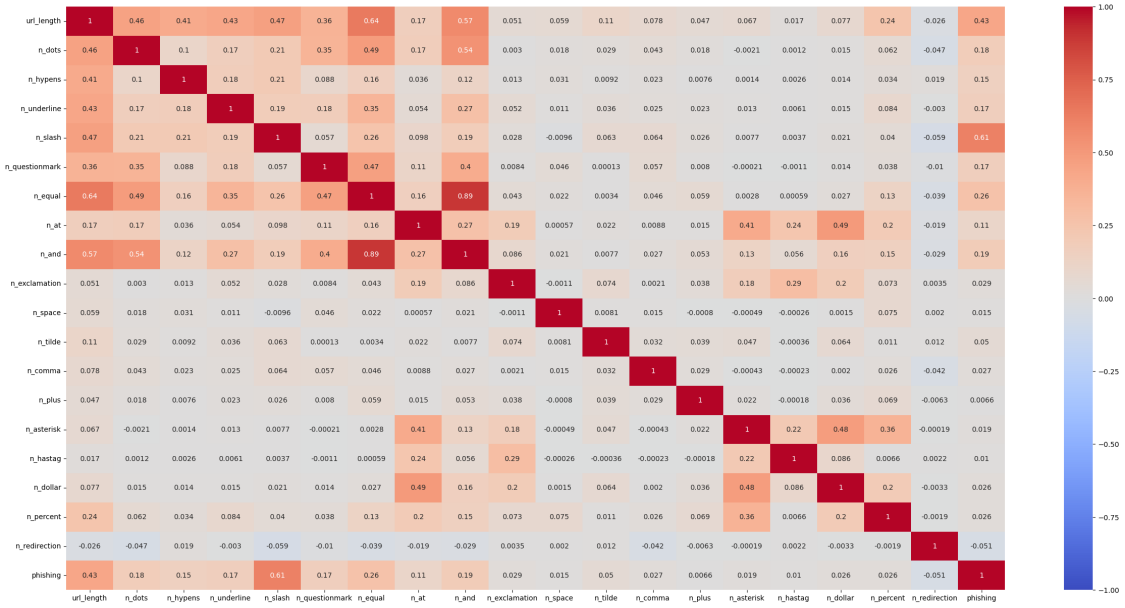
[4]:
```
data_phising = data.phishing.unique()
data
```

[4]:
```
        url_length  n_dots  n_hypens  n_underline  n_slash  n_questionmark  \
0               37       3         0            0        0               0
1               77       1         0            0        0               0
2              126       4         1            2        0               1
3               18       2         0            0        0               0
4               55       2         2            0        0               0
…              …      …        …          …       …              …
100072          23       3         1            0        0               0
100073          34       2         0            0        0               0
100074          70       2         1            0        5               0
```

```
100075          28       2         0            0        1                0
100076          16       2         0            0        0                0

        n_equal  n_at  n_and  n_exclamation  n_space  n_tilde  n_comma  \
0             0     0      0              0        0        0        0
1             0     0      0              0        0        0        0
2             3     0      2              0        0        0        0
3             0     0      0              0        0        0        0
4             0     0      0              0        0        0        0
…           …    …    …             …       …       …       …
100072        0     0      0              0        0        0        0
100073        0     0      0              0        0        0        0
100074        0     0      0              0        0        0        0
100075        0     0      0              0        0        0        0
100076        0     0      0              0        0        0        0

        n_plus  n_asterisk  n_hastag  n_dollar  n_percent  n_redirection  \
0            0           0         0         0          0              0
1            0           0         0         0          0              1
2            0           0         0         0          0              1
3            0           0         0         0          0              1
4            0           0         0         0          0              1
…          …         …       …       …         …            …
100072       0           0         0         0          0              0
100073       0           0         0         0          0              2
100074       0           0         0         0          0              0
100075       0           0         0         0          0              0
100076       0           0         0         0          0              0

        phishing
0              0
1              1
2              1
3              0
4              0
…            …
100072         0
100073         0
100074         1
100075         1
100076         0

[100077 rows x 20 columns]
```

[5]:
```
corr = data.corr()
plt.figure(figsize=(30,15))
```

```
sns.heatmap(corr, cmap='coolwarm', vmin=-1, vmax=1, annot=True)
plt.show()
```



[6]:
```
X = data.drop(['phishing'],axis=1).values
y = data['phishing'].values

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪3,random_state=1)
model = DecisionTreeClassifier(ccp_alpha= 0.0001, criterion= 'gini', max_depth=␣
↪32, min_samples_split= 5,random_state=1)
model.fit(X_train,y_train)

predictions_train = model.predict(X_train)
predictions_test = model.predict(X_test)

print(f"Train Accuracy Score: {accuracy_score(y_train,predictions_train)}")
print(f"Test Accuracy Score: {accuracy_score(y_test,predictions_test)}")
```

```
Train Accuracy Score: 0.8899547485475283
Test Accuracy Score: 0.8873567812416733
```

[7]:
```
# Adjust hyperparameters for better accuracy
model = GradientBoostingClassifier(n_estimators=500,  # Increase the number of␣
↪estimators
                                   learning_rate=0.1,  # Increase the learning␣
↪rate
                                   max_depth=8,        # Increase the max depth
```

```
                                    subsample=0.9,      # Increase the subsample
                                    max_features=0.7,   # Adjust max features
                                    random_state=42)

# Fit the model
model.fit(X_train, y_train)

# Make predictions
predictions_train = model.predict(X_train)
predictions_test = model.predict(X_test)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, predictions_train)
test_accuracy = accuracy_score(y_test, predictions_test)

print("Train Accuracy Score:", train_accuracy)
print("Test Accuracy Score:", test_accuracy)
```

```
Train Accuracy Score: 0.9219448132128532
Test Accuracy Score: 0.8946176392219558
```

[8]:
```
# Define your XGBoost classifier with adjusted hyperparameters
model = xgb.XGBClassifier(objective='binary:logistic',
                          colsample_bytree=0.8,
                          learning_rate=0.01,  # Adjusted learning rate
                          max_depth=6,         # Adjusted max depth
                          alpha=0.1,           # Adjusted regularization␣
 ↪parameter
                          n_estimators=500)    # Increased number of estimators

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on both training and testing data
predictions_train = model.predict(X_train)
predictions_test = model.predict(X_test)

# Calculate and print the accuracy scores
train_accuracy = accuracy_score(y_train, predictions_train)
test_accuracy = accuracy_score(y_test, predictions_test)

print("Train Accuracy Score:", train_accuracy)
print("Test Accuracy Score:", test_accuracy)
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
```

```
Train Accuracy Score: 0.8897120751431059
```

```
Test Accuracy Score: 0.8868904876099121
```

[9]:
```python
import numpy as np
import xgboost as xgb
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from deap import creator, base, tools, algorithms

# Define fitness function
def evaluate_fitness(individual):
    # Extract hyperparameters from the individual
    learning_rate, max_depth, alpha, n_estimators = individual

    # Create XGBoost classifier with given hyperparameters
    model = xgb.XGBClassifier(objective='binary:logistic',
                              colsample_bytree=0.8,
                              learning_rate=learning_rate,
                              max_depth=max_depth,
                              alpha=alpha,
                              n_estimators=n_estimators)

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

    # Make predictions
    predictions = model.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, predictions)

    return accuracy,

# Genetic Algorithm Parameters
pop_size = 50
cx_prob = 0.7  # Crossover probability
mut_prob = 0.2  # Mutation probability
num_generations = 10

# Create fitness function
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# Initialize toolbox
toolbox = base.Toolbox()

# Register parameter types and functions
toolbox.register("attr_float", np.random.uniform, 0.001, 0.1)  # Learning rate
```

```
toolbox.register("attr_int", np.random.randint, 1, 20)  # Max depth, alpha
toolbox.register("attr_int2", np.random.randint, 100, 500)  # Number of␣
 ↪estimators

toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_float, toolbox.attr_int, toolbox.attr_int,␣
 ↪toolbox.attr_int2), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=[1, 1, 1, 100], up=[20, 20,␣
 ↪20, 500], indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate_fitness)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize population
population = toolbox.population(n=pop_size)

# Run genetic algorithm
population, logbook = algorithms.eaSimple(population, toolbox, cxpb=cx_prob,␣
 ↪mutpb=mut_prob, ngen=num_generations, verbose=False)

# Get best individual from population
best_individual = tools.selBest(population, k=1)[0]

# Extract best hyperparameters
learning_rate, max_depth, alpha, n_estimators = best_individual

# Train the model with the best hyperparameters
model = xgb.XGBClassifier(objective='binary:logistic',
                          colsample_bytree=0.8,
                          learning_rate=learning_rate,
                          max_depth=max_depth,
                          alpha=alpha,
                          n_estimators=n_estimators)

model.fit(X_train, y_train)

# Make predictions
predictions_train = model.predict(X_train)
predictions_test = model.predict(X_test)

# Calculate accuracy scores
```

```
train_accuracy = accuracy_score(y_train, predictions_train)
test_accuracy = accuracy_score(y_test, predictions_test)

print("Best Hyperparameters:", best_individual)
print("Train Accuracy Score:", train_accuracy)
print("Test Accuracy Score:", test_accuracy)
```

```
Best Hyperparameters: [0.08931228199805653, 9, 1, 276]
Train Accuracy Score: 0.9081075679794157
Test Accuracy Score: 0.8966326938449241
```

[ ]: