Vulnerable JavaScript Functions Detection using Stacking of Convolutional Neural Networks (Supplementary Material)

Abdullah Sheneamer[1]

[1]**Department of Computer Science, Jazan University, Jazan, KSA**

Corresponding author:
Abdullah Sheneamer[1]

Email address: asheneamer@jazanu.edu.sa

ABSTRACT

Keywords:

DATA

- Ferenc et al's Datase Ferenc et al. (2019) `http://www.inf.u-szeged.hu/~ferenc/papers/JSVulnerabilityDataSet/`
- Viszkok et al's Dataset Viszkok et al. (2021) `https://security.snyk.io/`
- Apache Tomcat Dataset Ganesh et al. (2022) `https://github.com/palmafr/MDPIData2022/tree/main/datasets`
- Types of Vulnerabilities (Multi-Classes) Dataset Ganesh et al. (2021)

CODE

0.1 Machine learning for Binary classes

```
 # compare ensemble to each baseline classifier
 from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
from matplotlib import pyplot
import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import os
import matplotlib.pyplot as plt
#from skimage.transform import resize
import imread
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score,
confusion_matrix
import pickle
# loading library
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn import svm
from imread import imread, imsave
from PIL import Image
 import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Model
from keras.layers import Input
import seaborn as sns
from keras.layers.core import Activation, Dropout, Dense
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
import sys
import os
from math import log
import scipy as sp
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

import matplotlib.pyplot as plt
from keras.optimizers import SGD
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from numpy import unique
from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN, Embedding,
Reshape
from sklearn.metrics import confusion_matrix
from collections import Counter
from xgboost import XGBClassifier


df =  pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
JSVulnerabilityDataSet-1.0.csv')
#df =  pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
MDPIData2022-main/datasets/tomcat-final.csv')
#df = pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
MDPIData2022-main/datasets/struts-final.csv')



import matplotlib.pyplot as plt
df.columns


```

```
sns.countplot(x='Vuln', data=df)

X = df[['CC', 'CCL', 'CCO', 'CI', 'CLC', 'CLLC', 'McCC',
        'NL', 'NLE', 'CD', 'CLOC', 'DLOC', 'TCD', 'TCLOC',
        'LLOC', 'LOC', 'NOS', 'NUMPAR', 'TLLOC', 'TLOC',
        'TNOS', 'HOR_D', 'HOR_T', 'HON_D', 'HON_T',
        'HLEN', 'HVOC', 'HDIFF', 'HVOL', 'HEFF',
        'HBUGS', 'HTIME', 'CYCL', 'PARAMS',
        'CYCL_DENS']].values



#y = df['Vuln']
y=df['Vuln']
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
y = label_encoder.fit_transform(y)

counter = Counter(y)
print(counter)

#ros = RandomOverSampler(sampling_strategy={ 0: 10629,1: 10629},
random_state=42) # String
#X, y = ros.fit_resample(X, y)

#rus = RandomUnderSampler(sampling_strategy={ 0: 1496,1: 1496},
random_state=42) # String
#X, y = rus.fit_resample(X, y)

#smote = SMOTE() #SMOTE("minority")
#X, y= smote.fit_resample(X, y)



counter = Counter(y)
print(counter)


X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.33, shuffle=True, random_state=42, stratify=y)
X_test1=X_test
X_train1=X_train
y_test1=y_test
y_train1=y_train


from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)



from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

Xtrain_scaled = scaler.transform(X_train)
Xtest_scaled = scaler.transform(X_test)

Xtrain_scaled.shape
```

```
170
171
172
173 #X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.15,
174 random_state=42,stratify=y)
175 print('Splitted Successfully')
176
177
178 # get the dataset
179 #def get_dataset():
180 #        X, y = make_classification(n_samples=1000, n_features=20,
181 n_informative=15, n_redundant=5, random_state=1)
182 #        return X, y
183
184 # get a stacking ensemble of models
185 def get_stacking():
186     # define the base models
187     level0 = list()
188     level0.append(('cart', DecisionTreeClassifier()))
189     level0.append(('lr', LogisticRegression(solver='liblinear',
190 max_iter=1000)))
191     level0.append(('bayes', GaussianNB()))
192     level0.append(('xgboost', XGBClassifier()))
193     level0.append(('bagging', BaggingClassifier()))
194     level0.append(('RF', RandomForestClassifier()))
195     level0.append(('knn', KNeighborsClassifier()))
196     level0.append(('svm', SVC()))
197
198     # define meta learner model
199     level1 = LogisticRegression(solver='liblinear',max_iter=1000)
200     # define the stacking ensemble
201     model = StackingClassifier(estimators=level0, final_estimator=
202 level1, cv=10)
203     return model
204
205 # get a list of models to evaluate
206 def get_models():
207     models = dict()
208     models['cart'] = DecisionTreeClassifier()
209     models['lr'] = LogisticRegression(solver='liblinear',
210 max_iter=1000)
211     models['bayes'] = GaussianNB()
212     models['xgboost'] = XGBClassifier()
213     models['bagging']=BaggingClassifier()
214     models['rf']=RandomForestClassifier()
215     models['knn'] = KNeighborsClassifier()
216     models['svm'] = SVC()
217     models['stacking'] = get_stacking()
218
219
220     return models
221
222 from sklearn.metrics import classification_report, accuracy_score,
223 make_scorer
224
225 # evaluate a give model using cross-validation
226 def evaluate_model(model, X, y, name):
227     cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
228 random_state=1)
229     print("=========================================="+name+"==========
230 ==========================================")
```

```
        #scores = cross_val_score(model, X, y,
    scoring=make_scorer(classification_report_with_accuracy_score)
    , cv=cv, n_jobs=-1, error_score='raise')
        scores = cross_val_score(model, X, y, scoring="f1", cv=cv,
    n_jobs=-1, error_score='raise')
        return scores


# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
        #scores = evaluate_model(model, X, y, name)
        #results.append(scores)
        #names.append(name)
        print("==========================================="+name+"=========
    ==========================================")
        #print(name)
        cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
    random_state=42)
        #cvs1 = cross_val_score(model, X, y, scoring=
    make_scorer(classification_report_with_accuracy_score),
    cv=cv, n_jobs=-1, error_score='raise').mean()
        for score in ["roc_auc", "f1", "precision", "recall", "accuracy"]:
            cvs = cross_val_score(model, X, y, scoring=score, cv=cv,
        n_jobs=-1, error_score='raise').mean()
            print(score + " : " + str(cvs))
        #print(cvs1)
        print('\n')
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

0.2 Deep learning for Binary classes

```
 import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Model
from keras.layers import Input
import seaborn as sns
from keras.layers.core import Activation, Dropout, Dense
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
import sys
import os
from math import log
import scipy as sp
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

import matplotlib.pyplot as plt
from keras.optimizers import SGD
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
from sklearn.model_selection import train_test_split
```

```
from sklearn.datasets import load_iris
from numpy import unique
from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN,
Embedding, Reshape
from sklearn.metrics import confusion_matrix
from collections import Counter

df =  pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
JSVulnerabilityDataSet-1.0.csv')
import matplotlib.pyplot as plt
sns.countplot(x='Vuln', data=df)

X = df[[ 'CC', 'CCL', 'CCO', 'CI', 'CLC', 'CLLC', 'McCC',
         'NL', 'NLE', 'CD', 'CLOC', 'DLOC', 'TCD', 'TCLOC',
         'LLOC', 'LOC', 'NOS','NUMPAR', 'TLLOC', 'TLOC',
         'TNOS', 'HOR_D', 'HOR_T', 'HON_D', 'HON_T','HLEN',
         'HVOC', 'HDIFF', 'HVOL', 'HEFF', 'HBUGS', 'HTIME',
         'CYCL', 'PARAMS', 'CYCL_DENS']].values

y = df['Vuln']

y

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
y = label_encoder.fit_transform(y)

counter = Counter(y)
print(counter)

#ros = RandomOverSampler(sampling_strategy={ 0: 10629,1: 10629},
random_state=42) # String
#X, y = ros.fit_resample(X, y)

#rus = RandomUnderSampler(sampling_strategy={ 0: 1496,1: 1496},
random_state=42) # String
#X, y = rus.fit_resample(X, y)

#smote = SMOTE() #SMOTE("minority")
#X, y= smote.fit_resample(X, y)



counter = Counter(y)
print(counter)


X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.33,shuffle=True, random_state=42, stratify=y)
X_test1=X_test
X_train1=X_train
y_test1=y_test
y_train1=y_train


from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
354 from sklearn.preprocessing import MinMaxScaler
355 scaler = MinMaxScaler()
356 scaler.fit(X_train)
357
358 Xtrain_scaled = scaler.transform(X_train)
359 Xtest_scaled = scaler.transform(X_test)
360
361 Xtrain_scaled.shape
362
363
364 num_classes = 2
365
366 def model_VGG16(learning_rate=0.001, momentum=0.9):
367     model = Sequential()
368
369
370     model.add(Conv1D(64, 2, activation="relu", input_shape=(X.shape[1],1)))
371     model.add(Conv1D(64, 2, activation='relu',
372     kernel_initializer='he_uniform',padding='same'))
373     model.add(MaxPooling1D())
374
375     model.add(Conv1D(128,2, activation='relu',
376     kernel_initializer='he_uniform',padding='same'))
377     model.add(Conv1D(128, 2, activation='relu',
378     kernel_initializer='he_uniform',
379     padding='same'))
380     model.add(MaxPooling1D())
381
382     model.add(Conv1D(256, 2, activation='relu',
383     kernel_initializer='he_uniform',padding='same'))
384     model.add(Conv1D(256, 2, activation='relu',
385     kernel_initializer='he_uniform',
386     padding='same'))
387     model.add(MaxPooling1D())
388
389     model.add(Conv1D(512,2, activation='relu',
390     kernel_initializer='he_uniform',padding='same'))
391     model.add(Conv1D(512, 2, activation='relu',
392     kernel_initializer='he_uniform',
393     padding='same'))
394     model.add(Conv1D(512, 2, activation='relu',
395     kernel_initializer='he_uniform',
396     padding='same'))
397     model.add(MaxPooling1D())
398
399
400     model.add(Conv1D(512,2, activation='relu',
401     kernel_initializer='he_uniform',padding='same'))
402     model.add(Conv1D(512, 2, activation='relu',
403     kernel_initializer='he_uniform',
404     padding='same'))
405     model.add(Conv1D(512, 2, activation='relu',
406     kernel_initializer='he_uniform',
407     padding='same'))
408     model.add(MaxPooling1D())
409
410
411     model.add(Flatten())
412     #malware_model.add(Dropout(0.5))
413
414     model.add(Dense(4096, activation='relu',
```

```
415         kernel_initializer='he_uniform'))
416         model.add(Dropout(0.5))
417         model.add(Dense(4096, activation='relu',
418         kernel_initializer='he_uniform'))
419         model.add(Dropout(0.5))
420         model.add(Dense(1000, activation='relu',
421         kernel_initializer='he_uniform'))
422 
423         model.add(Dense(num_classes, activation='sigmoid'))
424 
425         # compile model
426         opt = SGD(lr=learning_rate, momentum=momentum)
427         #model.compil(optimizer=opt,
428         loss='categorical_crossentropy',
429         metrics=[tf.keras.metrics.Precision(),
430         tf.keras.metrics.Recall(), 'accuracy'])
431         model.compile(loss='binary_crossentropy',
432         optimizer=opt, metrics=['accuracy'])
433 
434         return model
435 
436 
437 
438 def model_VGG19(learning_rate=0.001, momentum=0.9):
439 
440 
441         model = Sequential()
442 
443         model.add(Conv1D(64, 2, activation="relu",
444         input_shape=(X.shape[1],1)))
445         model.add(Conv1D(64, 2, activation='relu',
446         kernel_initializer='he_uniform',
447         padding='same'))
448         model.add(MaxPooling1D())
449 
450         model.add(Conv1D(128,2, activation='relu',
451         kernel_initializer='he_uniform',padding='same'))
452         model.add(Conv1D(128, 2, activation='relu',
453         kernel_initializer='he_uniform', padding='same'))
454         model.add(MaxPooling1D())
455 
456         model.add(Conv1D(256, 2, activation='relu',
457         kernel_initializer='he_uniform', padding='same'))
458         model.add(Conv1D(256, 2, activation='relu',
459         kernel_initializer='he_uniform',
460         padding='same'))
461         model.add(MaxPooling1D())
462 
463         model.add(Conv1D(512,2, activation='relu',
464 
465         kernel_initializer='he_uniform',padding='same'))
466         model.add(Conv1D(512, 2, activation='relu',
467         kernel_initializer='he_uniform',padding='same'))
468         model.add(Conv1D(512, 2, activation='relu',
469         kernel_initializer='he_uniform',padding='same'))
470         model.add(Conv1D(512, 2, activation='relu',
471         kernel_initializer='he_uniform', padding='same'))
472         model.add(MaxPooling1D())
473 
474 
```

```
475         model.add(Conv1D(512,2, activation='relu',
476         kernel_initializer='he_uniform',padding='same'))
477         model.add(Conv1D(512, 2, activation='relu',
478         kernel_initializer='he_uniform', padding='same'))
479         model.add(Conv1D(512, 2, activation='relu',
480         kernel_initializer='he_uniform', padding='same'))
481         model.add(Conv1D(512, 2, activation='relu',
482         kernel_initializer='he_uniform',padding='same'))
483         model.add(MaxPooling1D())
484
485
486         model.add(Flatten())
487         #malware_model.add(Dropout(0.5))
488
489         model.add(Dense(4096, activation='relu',
490         kernel_initializer='he_uniform'))
491         model.add(Dropout(0.5))
492         model.add(Dense(4096, activation='relu',
493         kernel_initializer='he_uniform'))
494         model.add(Dropout(0.5))
495         model.add(Dense(1000, activation='relu',
496         kernel_initializer='he_uniform'))
497
498         model.add(Dense(num_classes, activation='sigmoid'))
499
500         # compile model
501         opt = SGD(lr=learning_rate, momentum=momentum)
502         #malware_model.compile(optimizer=opt,
503         loss='categorical_crossentropy',
504         metrics=[tf.keras.metrics.Precision(),
505         tf.keras.metrics.Recall(), 'accuracy'])
506         model.compile(loss='binary_crossentropy',
507         optimizer=opt, metrics=['accuracy'])
508
509         return model
510
511
512
513 def model_AlexNet(learning_rate=0.001, momentum=0.9):
514         model = Sequential()
515
516         model.add(Conv1D(96, 11, activation="relu",
517         input_shape=(X.shape[1],1)))
518         model.add(MaxPooling1D())
519
520         model.add(Conv1D(256, 2, activation='relu',
521         kernel_initializer='he_uniform',padding='same'))
522         model.add(MaxPooling1D())
523
524         model.add(Conv1D(384,2, activation='relu',
525         kernel_initializer='he_uniform',padding='same'))
526         model.add(Conv1D(384, 2, activation='relu',
527         kernel_initializer='he_uniform',padding='same'))
528         model.add(Conv1D(384, 2, activation='relu',
529         kernel_initializer='he_uniform', padding='same'))
530         model.add(MaxPooling1D())
531
532
533         model.add(Flatten())
534         model.add(Dense(4096, activation='relu',
```

```
535         kernel_initializer='he_uniform'))
536     model.add(Dropout(0.5))
537     model.add(Dense(4096, activation='relu',
538     kernel_initializer='he_uniform'))
539     model.add(Dropout(0.5))
540     model.add(Dense(1000, activation='relu',
541     kernel_initializer='he_uniform'))
542     model.add(Dense(num_classes, activation='sigmoid'))
543 
544     # compile model
545     opt = SGD(lr=learning_rate, momentum=momentum)
546     #malware_model.compile(optimizer=opt,
547     loss='categorical_crossentropy',
548     metrics=[tf.keras.metrics.Precision(),
549     tf.keras.metrics.Recall(), 'accuracy'])
550     model.compile(loss='binary_crossentropy',
551     optimizer=opt, metrics=['accuracy'])
552     return model
553 
554 
555 def model_Resent(learning_rate=0.001, momentum=0.9):
556     model = Sequential()
557 
558     model.add(Conv1D(64, 2, activation="relu",
559     input_shape=(X.shape[1],1)))
560     model.add(Dropout(0.5))
561     model.add(MaxPooling1D())
562 
563     model.add(Conv1D(64, 2, activation='relu',
564     kernel_initializer='he_uniform',padding='same'))
565     model.add(Conv1D(64, 2, activation='relu',
566     kernel_initializer='he_uniform',padding='same'))
567     model.add(Conv1D(64, 2, activation='relu',
568     kernel_initializer='he_uniform',padding='same'))
569     model.add(Conv1D(64, 2, activation='relu',
570     kernel_initializer='he_uniform',padding='same'))
571 
572     model.add(Conv1D(128,2, activation='relu',
573     kernel_initializer='he_uniform',padding='same'))
574     model.add(Dropout(0.5))
575 
576     model.add(Conv1D(128, 2, activation='relu',
577     kernel_initializer='he_uniform',padding='same'))
578     model.add(Conv1D(128,2, activation='relu',
579     kernel_initializer='he_uniform',padding='same'))
580     model.add(Conv1D(128, 2, activation='relu',
581     kernel_initializer='he_uniform',padding='same'))
582     model.add(Conv1D(128,2, activation='relu',
583     kernel_initializer='he_uniform',padding='same'))
584     model.add(Conv1D(128, 2, activation='relu',
585     kernel_initializer='he_uniform',padding='same'))
586     model.add(Conv1D(128,2, activation='relu',
587     kernel_initializer='he_uniform',padding='same'))
588     model.add(Conv1D(128, 2, activation='relu',
589     kernel_initializer='he_uniform',padding='same'))
590 
591     model.add(Conv1D(256, 2, activation='relu', kernel_initializer='he_uniform'
592     padding='same'))
593     model.add(Dropout(0.5))
```

```
594
595         model.add(Conv1D(256, 2, activation='relu',
596         kernel_initializer='he_uniform',padding='same'))
597         model.add(Conv1D(256, 2, activation='relu',
598         kernel_initializer='he_uniform',padding='same'))
599         model.add(Conv1D(256, 2, activation='relu',
600         kernel_initializer='he_uniform',padding='same'))
601         model.add(Conv1D(256, 2, activation='relu',
602         kernel_initializer='he_uniform',padding='same'))
603         model.add(Conv1D(256, 2, activation='relu',
604         kernel_initializer='he_uniform',padding='same'))
605         model.add(Conv1D(256, 2, activation='relu',
606         kernel_initializer='he_uniform',padding='same'))
607         model.add(Conv1D(256, 2, activation='relu',
608         kernel_initializer='he_uniform',padding='same'))
609         model.add(Conv1D(256, 2, activation='relu',
610         kernel_initializer='he_uniform',padding='same'))
611         model.add(Conv1D(256, 2, activation='relu',
612         kernel_initializer='he_uniform',padding='same'))
613         model.add(Conv1D(256, 2, activation='relu',
614         kernel_initializer='he_uniform',padding='same'))
615         model.add(Conv1D(256, 2, activation='relu',
616         kernel_initializer='he_uniform',padding='same'))
617
618         model.add(Conv1D(512,2, activation='relu',
619         kernel_initializer='he_uniform',padding='same'))
620         model.add(Dropout(0.5))
621
622         model.add(Conv1D(512, 2, activation='relu',
623         kernel_initializer='he_uniform',padding='same'))
624         model.add(Conv1D(512, 2, activation='relu',
625         kernel_initializer='he_uniform',padding='same'))
626         model.add(Conv1D(512, 2, activation='relu',
627         kernel_initializer='he_uniform',padding='same'))
628         model.add(Conv1D(512,2, activation='relu',
629         kernel_initializer='he_uniform',padding='same'))
630         model.add(Conv1D(512, 2, activation='relu',
631         kernel_initializer='he_uniform',padding='same'))
632
633         model.add(MaxPooling1D())
634
635
636         model.add(Flatten())
637         model.add(Dropout(0.5))
638
639         model.add(Dense(1000, activation='relu',
640         kernel_initializer='he_uniform'))
641
642         model.add(Dense(num_classes, activation='sigmoid'))
643
644         # compile model
645         opt = SGD(lr=learning_rate, momentum=momentum)
646         #malware_model.compile(optimizer=opt,
647         loss='categorical_crossentropy',
648         metrics=[tf.keras.metrics.Precision(),
649         tf.keras.metrics.Recall(), 'accuracy'])
650         model.compile(loss='binary_crossentropy',
651         optimizer=opt, metrics=['accuracy'])
652
```

```
    return model

def model_LSTM(learning_rate=0.001, momentum=0.9):

    input_layer = Input(shape=(X.shape[1],1))

    conv1 = Conv1D(filters=35,
                kernel_size=8,
                strides=1,
                activation='relu')(input_layer)
    pool1 = MaxPooling1D(pool_size=4)(conv1)
    lstm1 = LSTM(35)(pool1)
    output_layer = Dense(2, activation='sigmoid')(lstm1)
    model = Model(inputs=input_layer, outputs=output_layer)
    opt = SGD(lr=learning_rate, momentum=momentum)
    model.compile(loss='binary_crossentropy',
    optimizer=opt, metrics=['acc'])

    return model



print(unique(y_test))
Vuln_model1 = model_VGG16()
Vuln_model2 = model_VGG19()
Vuln_model3 = model_AlexNet()
Vuln_model4 = model_Resent()
Vuln_model5 = model_LSTM()




y_train_new = np.argmax(y_train, axis=1)

y_train_new

from sklearn.utils import class_weight
from sklearn.utils import compute_class_weight

class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y_train_new),
                                        y = y_train_new
                                    )
class_weights = dict(zip(np.unique(y_train_new), class_weights))
class_weights




history1 = Vuln_model1.fit(x= Xtrain_scaled,
y=y_train, batch_size=128, epochs=1000, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores1 = Vuln_model1.evaluate(x= Xtest_scaled,
y=y_test, verbose=1)
print('Final Vuln_model1 (VGG16) accuracy: ', scores1[1])

plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])

plt.title('VGG16 model accuracy')
plt.ylabel('accuracy')
```

```
715 plt.xlabel('epoch')
716 plt.legend(['train','test'], loc='upper left')
717 plt.show()
718
719 plt.plot(history1.history['loss'])
720 plt.plot(history1.history['val_loss'])
721
722 plt.title('VGG16 model loss')
723 plt.ylabel('loss')
724 plt.xlabel('epoch')
725 plt.legend(['train','test'], loc='upper left')
726 plt.show()
727
728
729 classes = Vuln_model1.predict(x= Xtest_scaled)
730 y_classes = classes.argmax(axis=-1)
731 true_classes = y_test.argmax(axis=-1)
732
733 print(classification_report(true_classes, y_classes,digits=6))
734
735 cf_matrix = confusion_matrix(true_classes, y_classes)
736 print(cf_matrix)
737
738
739
740 history2 = Vuln_model2.fit(x= Xtrain_scaled,
741 y=y_train, batch_size=128, epochs=1000, verbose=1,
742 validation_split=0.33,class_weight=class_weights)
743 scores2 = Vuln_model2.evaluate(x= Xtest_scaled,
744 y=y_test, verbose=1)
745 print('Final Vuln_model2 (VGG19) accuracy: ', scores1[1])
746
747 import matplotlib.pyplot as plt2
748
749
750 print("Test Score:", scores2[0])
751 print("Test Accuracy:", scores2[1])
752
753 plt2.plot(history2.history['accuracy'])
754 plt2.plot(history2.history['val_accuracy'])
755
756 plt2.title('VGG19 model accuracy')
757 plt2.ylabel('accuracy')
758 plt2.xlabel('epoch')
759 plt2.legend(['train','test'], loc='upper left')
760 plt2.show()
761
762 plt2.plot(history2.history['loss'])
763 plt2.plot(history2.history['val_loss'])
764
765 plt2.title('VGG19 model loss')
766 plt2.ylabel('loss')
767 plt2.xlabel('epoch')
768 plt2.legend(['train','test'], loc='upper left')
769 plt2.show()
770
771
772 classes = Vuln_model2.predict(x= Xtest_scaled)
773 y_classes = classes.argmax(axis=-1)
774 true_classes = y_test.argmax(axis=-1)
```

```
775
776 print(classification_report(true_classes, y_classes,digits=6))
777
778
779 cf_matrix = confusion_matrix(true_classes, y_classes)
780 print(cf_matrix)
781
782 '''
783 group_names1 = ['True Neg','False Pos','False Neg','True Pos']
784 group_counts1 = ["{0:0.0f}".format(value) for value in
785                  cf_matrix.flatten()]
786 group_percentages1 = ["{0:.2%}".format(value) for value in
787                       cf_matrix.flatten()/np.sum(cf_matrix)]
788 labels1 = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
789           zip(group_names1,group_counts1,group_percentages1)]
790 labels1 = np.asarray(labels1).reshape(2,2)
791 sns.heatmap(cf_matrix, annot=labels1, fmt='', cmap='Blues')
792 '''
793
794
795
796 import matplotlib.pyplot as plt3
797
798
799 history3 = Vuln_model3.fit(x= Xtrain_scaled,
800 y=y_train, batch_size=128, epochs=1000, verbose=1,
801 validation_split=0.33,class_weight=class_weights)
802 scores3 = Vuln_model3.evaluate(x= Xtest_scaled,
803 y=y_test, verbose=1)
804 print('Final Vuln_model3 (AlexNet) accuracy: ', scores3[1])
805
806
807 print("Test Score:", scores3[0])
808 print("Test Accuracy:", scores3[1])
809
810 plt3.plot(history3.history['accuracy'])
811 plt3.plot(history3.history['val_accuracy'])
812
813 plt3.title('AlexNet model accuracy')
814 plt3.ylabel('accuracy')
815 plt3.xlabel('epoch')
816 plt3.legend(['train','test'], loc='upper left')
817 plt3.show()
818
819 plt3.plot(history3.history['loss'])
820 plt3.plot(history3.history['val_loss'])
821
822 plt3.title('AlexNet model loss')
823 plt3.ylabel('loss')
824 plt3.xlabel('epoch')
825 plt3.legend(['train','test'], loc='upper left')
826 plt3.show()
827
828
829 classes = Vuln_model3.predict(x= Xtest_scaled)
830 y_classes = classes.argmax(axis=-1)
831 true_classes = y_test.argmax(axis=-1)
832 print(classification_report(true_classes, y_classes,digits=6))
833
834
835
```

```
cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

import matplotlib.pyplot as plt4

history4 = Vuln_model4.fit(x= Xtrain_scaled,
y=y_train, batch_size=128, epochs=1000, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores4 = Vuln_model4.evaluate(x= Xtest_scaled,
y=y_test, verbose=1)
print('Final Vuln_model4 (Resent) accuracy: ', scores4[1])


print("Test Score:", scores4[0])
print("Test Accuracy:", scores4[1])

plt4.plot(history4.history['accuracy'])
plt4.plot(history4.history['val_accuracy'])

plt4.title('Resent model accuracy')
plt4.ylabel('accuracy')
plt4.xlabel('epoch')
plt4.legend(['train','test'], loc='upper left')
plt4.show()

plt4.plot(history4.history['loss'])
plt4.plot(history4.history['val_loss'])

plt4.title('Resent model loss')
plt4.ylabel('loss')
plt4.xlabel('epoch')
plt4.legend(['train','test'], loc='upper left')
plt4.show()


classes = Vuln_model4.predict(x= Xtest_scaled)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)
print(classification_report(true_classes, y_classes,digits=6))

cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

history5 = Vuln_model5.fit(x= X_train, y=y_train,
batch_size=128, epochs=1000, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores5 = Vuln_model5.evaluate(x= X_test,
y=y_test, verbose=1)
print('Final Vuln_model5 (LSTM) accuracy: ', scores5[1])

import matplotlib.pyplot as plt5
print("Test Score:", scores5[0])
print("Test Accuracy:", scores5[1])

plt5.plot(history5.history['acc'])
plt5.plot(history5.history['val_acc'])

plt5.title('LSTM model accuracy')
plt5.ylabel('accuracy')
plt5.xlabel('epoch')
```

```
plt5.legend(['train','test'], loc='upper left')
plt5.show()

plt5.plot(history5.history['loss'])
plt5.plot(history5.history['val_loss'])

plt5.title('LSTM model loss')
plt5.ylabel('loss')
plt5.xlabel('epoch')
plt5.legend(['train','test'], loc='upper left')
plt5.show()


classes = Vuln_model5.predict(x= X_test)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)

print(classification_report(true_classes, y_classes,digits=6))



cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')



Vuln_model1.save('/Users/abdullah/models/Model1.h5')
Vuln_model2.save('/Users/abdullah/models/Model2.h5')
Vuln_model3.save('/Users/abdullah/models/Model3.h5')
Vuln_model4.save('/Users/abdullah/models/Model4.h5')
Vuln_model5.save('/Users/abdullah/models/Model5.h5')



from keras.models import load_model

# load models from file
def load_all_models(n_models):
        all_models = list()
        for i in range(n_models):
                # define filename for this ensemble
                filename = '/Users/abdullah/models/Model' + str(i+1) + '.h5'
                # load model from file
                model = load_model(filename)
                # add to list of members
                all_models.append(model)
                print('>loaded \%s' \% filename)
        return all_models


```

```
# load all models
n_members = 5
members = load_all_models(n_members)
print('Loaded %d models' % len(members))


 # create stacked model input dataset as outputs from the ensemble
def stacked_dataset(members, inputX):
        stackX = None
        for model in members:
                # make prediction
                yhat = model.predict(inputX, verbose=0)
                # stack predictions into [rows, members, probabilities]
                if stackX is None:
                        stackX = yhat
                else:
                        stackX = dstack((stackX, yhat))
        # flatten predictions to [rows, members x probabilities]
        stackX = stackX.reshape((stackX.shape[0],
              stackX.shape[1]*stackX.shape[2]))
        return stackX



 # fit a model based on the outputs from the ensemble members
def fit_stacked_model(members, inputX, inputy):
        # create dataset using ensemble
        stackedX = stacked_dataset(members, inputX)
        # fit standalone model
        model = LogisticRegression()
        model.fit(stackedX, inputy)
        return model


# fit stacked model using the ensemble
from numpy import dstack
from sklearn.linear_model import LogisticRegression
y_test=np.argmax(y_test, axis=1)

model = fit_stacked_model(members, Xtest_scaled, y_test)



# make a prediction with the stacked model

def stacked_prediction(members, model, inputX):
        # create dataset using ensemble
        stackedX = stacked_dataset(members, inputX)
        # make a prediction
        yhat = model.predict(stackedX)
        return yhat

# evaluate model on test set
from sklearn.metrics import accuracy_score
yhat = stacked_prediction(members, model, Xtest_scaled)

acc = accuracy_score(y_test, yhat)
print('Stacked Test Accuracy: %.3f' % acc)
print(len(yhat))


print(classification_report(y_test, yhat,digits=6))
```

```
cf_matrix = confusion_matrix(y_test, yhat)
print(cf_matrix)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2\%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
}
```

0.3 Machine learning for Multi-classes

```
 # compare ensemble to each baseline classifier
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
from matplotlib import pyplot
import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import os
import matplotlib.pyplot as plt
#from skimage.transform import resize
import imread
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import
classification_report,accuracy_score,confusion_matrix
import pickle
# loading library
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn import svm
from imread import imread, imsave
from PIL import Image
 import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Model
from keras.layers import Input
import seaborn as sns
from keras.layers.core import Activation, Dropout, Dense
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import classification_report
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
import sys
import os
from math import log
import scipy as sp
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

import matplotlib.pyplot as plt
from keras.optimizers import SGD
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from numpy import unique
from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN, Embedding, Res
from sklearn.metrics import confusion_matrix
from collections import Counter
from xgboost import XGBClassifier
import matplotlib.pyplot as plt

df =pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/
Code_Metrics_Datasets/Multi-Classes/PU_Dataset.csv')
df1 =pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/
Code_Metrics_Datasets/Multi-Classes/cwe119_GCDFile.csv')
df2 =pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
Vulnerability_Research/Vul_Datasets/
Code_Metrics_Datasets/Multi-Classes/cwe399_cgd.csv')
df3 =pd.read_csv('/Users/abdullah/Desktop/Folder
/Research/Vulnerability_Research/Vul_Datasets/
Code_Metrics_Datasets/Multi-Classes/API_function_call.csv')
df4 =pd.read_csv('/Users/abdullah/Desktop/Folder
/Research/Vulnerability_Research/Vul_Datasets/
Code_Metrics_Datasets/Multi-Classes/Array_usage.csv')
df5 =pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/
Code_Metrics_Datasets/Multi-Classes/Arithmetic_expression.csv')

import matplotlib.pyplot as plt
df.columns


sns.countplot(x='IsVulnerable', data=df)

X = df[['Lines_Of_Program','Physic_Lines',
        'n1_Number_Of_Distinct_Operators',
        'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
        'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
        'N_Program_Length','B_Number_of_Delivered_Bugs_1',
        'B_Number_of_Delivered_Bugs_2',
        'D_Difficulty','E_Effort','T_Time_Required_To_Program',
        'V_Volume','_N_Calculated_Program_Length',
        'McCab_Number']].values


y= df['IsVulnerable'].values

```

```
1145
1146
1147 print(len(y))
1148
1149
1150
1151
1152 import matplotlib.pyplot as plt1
1153 df1.columns
1154
1155
1156 sns.countplot(x='IsVulnerable', data=df1)
1157
1158 X1 = df1[['Lines_Of_Program','Physic_Lines',
1159          'n1_Number_Of_Distinct_Operators',
1160          'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
1161          'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
1162          'N_Program_Length','B_Number_of_Delivered_Bugs_1',
1163          'B_Number_of_Delivered_Bugs_2','D_Difficulty','E_Effort',
1164          'T_Time_Required_To_Program','V_Volume',
1165          '_N_Calculated_Program_Length',
1166          'McCab_Number']].values
1167
1168 y1= df1['IsVulnerable'].values
1169
1170
1171 print(len(y1))
1172
1173
1174 X2 = df2[['Lines_Of_Program','Physic_Lines',
1175          'n1_Number_Of_Distinct_Operators',
1176          'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
1177          'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
1178          'N_Program_Length','B_Number_of_Delivered_Bugs_1',
1179          'B_Number_of_Delivered_Bugs_2','D_Difficulty',
1180          'E_Effort','T_Time_Required_To_Program','V_Volume',
1181          '_N_Calculated_Program_Length',
1182          'McCab_Number']].values
1183
1184
1185 y2= df2['IsVulnerable'].values
1186
1187
1188
1189
1190
1191
1192 print(len(y2))
1193
1194
1195
1196 X3 = df3[['Lines_Of_Program','Physic_Lines',
1197          'n1_Number_Of_Distinct_Operators','n2_Number_Of_Distinct_Operands',
1198          'n_Program_Vocabulary','N1_Total_Number_Of_Operators',
1199          'N2_Total_Number_Of_Operands','N_Program_Length',
1200          'B_Number_of_Delivered_Bugs_1','B_Number_of_Delivered_Bugs_2',
1201          'D_Difficulty','E_Effort','T_Time_Required_To_Program','V_Volume',
1202          '_N_Calculated_Program_Length','McCab_Number']].values
1203
1204 y3= df3['IsVulnerable'].values
1205
1206 print(len(y3))
1207
1208
```

```
X4 = df4[['Lines_Of_Program','Physic_Lines',
          'n1_Number_Of_Distinct_Operators',
          'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
          'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
          'N_Program_Length','B_Number_of_Delivered_Bugs_1',
          'B_Number_of_Delivered_Bugs_2','D_Difficulty',
          'E_Effort','T_Time_Required_To_Program','V_Volume',
          '_N_Calculated_Program_Length','McCab_Number']].values


y4= df4['IsVulnerable'].values

X5 = df5[['Lines_Of_Program','Physic_Lines',
         'n1_Number_Of_Distinct_Operators',
         'n2_Number_Of_Distinct_Operands',
         'n_Program_Vocabulary','N1_Total_Number_Of_Operators',
         'N2_Total_Number_Of_Operands','N_Program_Length',
         'B_Number_of_Delivered_Bugs_1','B_Number_of_Delivered_Bugs_2',
         'D_Difficulty','E_Effort','T_Time_Required_To_Program',
         'V_Volume','_N_Calculated_Program_Length',
         'McCab_Number']].values


y5= df5['IsVulnerable'].values


print(len(y5))


import numpy as np
X=np.concatenate((X,X1,X2,X3,X4,X5))
y=np.concatenate((y,y1,y2,y3,y4,y5))

import numpy
print(numpy.unique(y))

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
y = label_encoder.fit_transform(y)
print(numpy.unique(y))

#from sklearn.preprocessing import label_binarize
#y = label_binarize(y, classes=[0, 1, 2, 3,4,5,6])

#counter = Counter(y)
#print(counter)

X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.33,shuffle=True, random_state=42, stratify=y)
X_test1=X_test
X_train1=X_train
y_test1=y_test
y_train1=y_train




from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
1272
1273
1274
1275 from sklearn.preprocessing import MinMaxScaler
1276 scaler = MinMaxScaler()
1277 scaler.fit(X_train)
1278
1279 Xtrain_scaled = scaler.transform(X_train)
1280 Xtest_scaled = scaler.transform(X_test)
1281
1282 Xtrain_scaled.shape
1283
1284
1285
1286 #X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.15,
1287 random_state=42,stratify=y)
1288 print('Splitted Successfully')
1289
1290
1291 # get the dataset
1292 #def get_dataset():
1293 #       X, y = make_classification(n_samples=1000, n_features=20,
1294 n_informative=15, n_redundant=5, random_state=1)
1295 #       return X, y
1296
1297 # get a stacking ensemble of models
1298 def get_stacking():
1299         # define the base models
1300         level0 = list()
1301         level0.append(('cart', DecisionTreeClassifier()))
1302         level0.append(('lr', LogisticRegression(solver='liblinear',
1303     max_iter=1000)))
1304         level0.append(('bayes', GaussianNB()))
1305         level0.append(('xgboost', XGBClassifier()))
1306         level0.append(('bagging', BaggingClassifier()))
1307         level0.append(('RF', RandomForestClassifier()))
1308         level0.append(('knn', KNeighborsClassifier()))
1309         level0.append(('svm', SVC()))
1310
1311         # define meta learner model
1312         level1 = LogisticRegression(solver='liblinear',max_iter=1000)
1313         # define the stacking ensemble
1314         model = StackingClassifier(estimators=level0,
1315     final_estimator=level1, cv=10)
1316     return model
1317
1318 # get a list of models to evaluate
1319 def get_models():
1320         models = dict()
1321         models['cart'] = DecisionTreeClassifier()
1322         models['lr'] = LogisticRegression(solver='liblinear',max_iter=1000)
1323         models['bayes'] = GaussianNB()
1324         models['xgboost'] = XGBClassifier()
1325         models['bagging']=BaggingClassifier()
1326         models['rf']=RandomForestClassifier()
1327         models['knn'] = KNeighborsClassifier()
1328         models['svm'] = SVC()
1329         models['stacking'] = get_stacking()
1330
1331
1332         return models
```

```
from sklearn.metrics import classification_report,
accuracy_score, make_scorer

# evaluate a give model using cross-validation
def evaluate_model(model, X, y,name):
        cv = RepeatedStratifiedKFold(n_splits=10,
    n_repeats=3, random_state=1)
        print("======================================"+name+"=====
    ==========================================")
        #scores = cross_val_score(model, X, y, scoring=make_scorer(
              classification_report_with_accuracy_score),
                cv=cv, n_jobs=-1, error_score='raise')
        scores = cross_val_score(model, X, y, scoring="f1",
              cv=cv, n_jobs=-1, error_score='raise')
        return scores


# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
        #scores = evaluate_model(model, X, y,name)
        #results.append(scores)
        #names.append(name)
        print("======================================"+name+"========
    ============================================")
        #print(name)
        cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
          random_state=42)
        for score in ["roc_auc", "f1", "precision",
                  "recall", "accuracy"]:
                cvs = cross_val_score(model, X, y,
                                      scoring=score, cv=cv, n_jobs=-1,
                                      error_score='raise').mean()
                print(score + " : "+ str(cvs))
        #print(cvs1)
        print('\n')
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

0.4 Deep learning for Multi-classes

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Model
from keras.layers import Input
import seaborn as sns
from keras.layers.core import Activation, Dropout, Dense
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
import sys
import os
from math import log
```

```
import scipy as sp
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

import matplotlib.pyplot as plt
from keras.optimizers import SGD
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from numpy import unique
from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN,
Embedding, Reshape
from sklearn.metrics import confusion_matrix
from collections import Counter

df =  pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
Multi-Classes/PU_Dataset.csv')
df1 =  pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
Multi-Classes/cwe119_GCDFile.csv')
df2 =  pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
Multi-Classes/cwe399_cgd.csv')
df3 =  pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
Multi-Classes/API_function_call.csv')
df4 =  pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
Multi-Classes/Array_usage.csv')
df5 =  pd.read_csv('/Users/abdullah/Desktop/Folder/
Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
Multi-Classes/Arithmetic_expression.csv')




import matplotlib.pyplot as plt
df.columns


sns.countplot(x='IsVulnerable', data=df)

X = df[['Lines_Of_Program','Physic_Lines',
        'n1_Number_Of_Distinct_Operators',
        'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
        'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
        'N_Program_Length','B_Number_of_Delivered_Bugs_1',
        'B_Number_of_Delivered_Bugs_2',
        'D_Difficulty','E_Effort','T_Time_Required_To_Program',
        'V_Volume','_N_Calculated_Program_Length',
        'McCab_Number']].values


y= df['IsVulnerable'].values



print(len(y))
```

```
1455
1456
1457
1458
1459 import matplotlib.pyplot as plt1
1460 df1.columns
1461
1462
1463 sns.countplot(x='IsVulnerable', data=df1)
1464
1465 X1 = df1[['Lines_Of_Program','Physic_Lines',
1466           'n1_Number_Of_Distinct_Operators',
1467           'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
1468           'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
1469           'N_Program_Length','B_Number_of_Delivered_Bugs_1',
1470           'B_Number_of_Delivered_Bugs_2','D_Difficulty',
1471           'E_Effort','T_Time_Required_To_Program','V_Volume',
1472           '_N_Calculated_Program_Length','McCab_Number']].values
1473
1474
1475 y1= df1['IsVulnerable'].values
1476
1477
1478 print(len(y1))
1479
1480
1481
1482
1483 X2 = df2[['Lines_Of_Program','Physic_Lines',
1484           'n1_Number_Of_Distinct_Operators',
1485           'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
1486           'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
1487           'N_Program_Length','B_Number_of_Delivered_Bugs_1',
1488           'B_Number_of_Delivered_Bugs_2', 'D_Difficulty','E_Effort',
1489           'T_Time_Required_To_Program','V_Volume',
1490           '_N_Calculated_Program_Length','McCab_Number']].values
1491
1492
1493 y2= df2['IsVulnerable'].values
1494
1495
1496
1497
1498
1499
1500 print(len(y2))
1501
1502
1503
1504 X3 = df3[['Lines_Of_Program','Physic_Lines',
1505           'n1_Number_Of_Distinct_Operators',
1506           'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
1507           'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
1508           'N_Program_Length','B_Number_of_Delivered_Bugs_1',
1509           'B_Number_of_Delivered_Bugs_2','D_Difficulty','E_Effort',
1510           'T_Time_Required_To_Program','V_Volume',
1511           '_N_Calculated_Program_Length','McCab_Number']].values
1512
1513
1514
1515 y3= df3['IsVulnerable'].values
1516
1517
1518 print(len(y3))
```

```
X4 = df4[['Lines_Of_Program','Physic_Lines',
          'n1_Number_Of_Distinct_Operators',
          'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
          'N1_Total_Number_Of_Operators','N2_Total_Number_Of_Operands',
          'N_Program_Length','B_Number_of_Delivered_Bugs_1',
          'B_Number_of_Delivered_Bugs_2',
          'D_Difficulty','E_Effort',
          'T_Time_Required_To_Program','V_Volume',
          '_N_Calculated_Program_Length','McCab_Number']].values


y4= df4['IsVulnerable'].values




X5 = df5[['Lines_Of_Program','Physic_Lines',
          'n1_Number_Of_Distinct_Operators',
          'n2_Number_Of_Distinct_Operands','n_Program_Vocabulary',
          'N1_Total_Number_Of_Operators',
          'N2_Total_Number_Of_Operands','N_Program_Length',
          'B_Number_of_Delivered_Bugs_1',
          'B_Number_of_Delivered_Bugs_2',
          'D_Difficulty','E_Effort',
          'T_Time_Required_To_Program','V_Volume',
          '_N_Calculated_Program_Length','McCab_Number']].values


y5= df5['IsVulnerable'].values


print(len(y5))


import numpy as np
X=np.concatenate((X,X1,X2,X3,X4,X5))
y=np.concatenate((y,y1,y2,y3,y4,y5))



# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
y = label_encoder.fit_transform(y)

counter = Counter(y)
print(counter)

#ros = RandomOverSampler(sampling_strategy={ 0: 85399,1: 85399},
random_state=42) # String
#X, y = ros.fit_resample(X, y)

#rus = RandomUnderSampler(sampling_strategy={ 0: 3475,1:
3475,2:3475,3:3475,4:3475,5:3475,6:3475}, random_state=42) # String
#X, y = rus.fit_resample(X, y)
```

```
#smote = SMOTE() #SMOTE("minority")
#X, y= smote.fit_resample(X, y)

rus = RandomUnderSampler(sampling_strategy={0:15000,1:
9952,3:13603,2:10440,4:7285,5:3475,6:10926}, random_state=42) # String
X, y = rus.fit_resample(X, y)



X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.33,shuffle=True, random_state=42, stratify=y)
X_test1=X_test
X_train1=X_train
y_test1=y_test
y_train1=y_train


from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)



from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

Xtrain_scaled = scaler.transform(X_train)
Xtest_scaled = scaler.transform(X_test)

Xtrain_scaled.shape


num_classes = 7

def model_VGG16(learning_rate=0.001, momentum=0.9):
    model = Sequential()


    model.add(Conv1D(64, 2, activation="relu",
    input_shape=(X.shape[1],1)))
    model.add(Conv1D(64, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(MaxPooling1D(pool_size=1))

    model.add(Conv1D(128,2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(128, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(MaxPooling1D(pool_size=1))

    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(MaxPooling1D(pool_size=1))

    model.add(Conv1D(512,2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(512, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
```

```
1643        model.add(Conv1D(512, 2, activation='relu',
1644        kernel_initializer='he_uniform',padding='same'))
1645        model.add(MaxPooling1D(pool_size=1))
1646
1647
1648        model.add(Conv1D(512,2, activation='relu',
1649        kernel_initializer='he_uniform',padding='same'))
1650        model.add(Conv1D(512, 2, activation='relu',
1651        kernel_initializer='he_uniform',padding='same'))
1652        model.add(Conv1D(512, 2, activation='relu',
1653        kernel_initializer='he_uniform',padding='same'))
1654        model.add(MaxPooling1D(pool_size=1))
1655
1656
1657        model.add(Flatten())
1658        #malware_model.add(Dropout(0.5))
1659
1660        model.add(Dense(4096, activation='relu',
1661        kernel_initializer='he_uniform'))
1662        model.add(Dropout(0.5))
1663        model.add(Dense(4096, activation='relu',
1664        kernel_initializer='he_uniform'))
1665        model.add(Dropout(0.5))
1666        model.add(Dense(1000, activation='relu',
1667        kernel_initializer='he_uniform'))
1668
1669        model.add(Dense(num_classes, activation='softmax'))
1670
1671        # compile model
1672        opt = SGD(lr=learning_rate, momentum=momentum)
1673        #malware_model.compile(optimizer=opt,
1674        loss='categorical_crossentropy',
1675        metrics=[tf.keras.metrics.Precision(),
1676        tf.keras.metrics.Recall(), 'accuracy'])
1677        model.compile(loss='categorical_crossentropy',
1678        optimizer=opt, metrics=['accuracy'])
1679
1680        return model
1681
1682
1683
1684    def model_VGG19(learning_rate=0.01, momentum=0.9):
1685
1686
1687        model = Sequential()
1688
1689        model.add(Conv1D(64, 2, activation="relu",
1690        input_shape=(X.shape[1],1)))
1691        model.add(Conv1D(64, 2, activation='relu',
1692        kernel_initializer='he_uniform',padding='same'))
1693        model.add(MaxPooling1D(pool_size=1))
1694
1695        model.add(Conv1D(128,2, activation='relu',
1696        kernel_initializer='he_uniform',padding='same'))
1697        model.add(Conv1D(128, 2, activation='relu',
1698        kernel_initializer='he_uniform',padding='same'))
1699        model.add(MaxPooling1D(pool_size=1))
1700
1701        model.add(Conv1D(256, 2, activation='relu',
1702        kernel_initializer='he_uniform',padding='same'))
```

```
1703        model.add(Conv1D(256, 2, activation='relu',
1704        kernel_initializer='he_uniform',padding='same'))
1705        model.add(MaxPooling1D(pool_size=1))
1706
1707        model.add(Conv1D(512,2, activation='relu',
1708        kernel_initializer='he_uniform', padding='same'))
1709        model.add(Conv1D(512, 2, activation='relu',
1710        kernel_initializer='he_uniform',padding='same'))
1711        model.add(Conv1D(512, 2, activation='relu',
1712        kernel_initializer='he_uniform',padding='same'))
1713        model.add(Conv1D(512, 2, activation='relu',
1714        kernel_initializer='he_uniform',padding='same'))
1715        model.add(MaxPooling1D(pool_size=1))
1716
1717
1718        model.add(Conv1D(512,2, activation='relu',
1719        kernel_initializer='he_uniform',padding='same'))
1720        model.add(Conv1D(512, 2, activation='relu',
1721        kernel_initializer='he_uniform',padding='same'))
1722        model.add(Conv1D(512, 2, activation='relu',
1723        kernel_initializer='he_uniform',
1724        padding='same'))
1725        model.add(Conv1D(512, 2, activation='relu',
1726        kernel_initializer='he_uniform',padding='same'))
1727        model.add(MaxPooling1D(pool_size=1))
1728
1729
1730        model.add(Flatten())
1731        #malware_model.add(Dropout(0.5))
1732
1733        model.add(Dense(4096, activation='relu',
1734        kernel_initializer='he_uniform'))
1735        model.add(Dropout(0.5))
1736        model.add(Dense(4096, activation='relu',
1737        kernel_initializer='he_uniform'))
1738        model.add(Dropout(0.5))
1739        model.add(Dense(1000, activation='relu',
1740        kernel_initializer='he_uniform'))
1741
1742        model.add(Dense(num_classes, activation='softmax'))
1743
1744        # compile model
1745        opt = SGD(lr=learning_rate, momentum=momentum)
1746        #malware_model.compile(optimizer=opt,
1747        loss='categorical_crossentropy',
1748        metrics=[tf.keras.metrics.Precision(),
1749        tf.keras.metrics.Recall(), 'accuracy'])
1750        model.compile(loss='categorical_crossentropy',
1751        optimizer=opt, metrics=['accuracy'])
1752
1753        return model
1754
1755
1756
1757  def model_AlexNet(learning_rate=0.01, momentum=0.9):
1758        model = Sequential()
1759
1760        model.add(Conv1D(96, 11, activation="relu",
1761        input_shape=(X.shape[1],1)))
1762        model.add(MaxPooling1D(pool_size=1))
```

```
1763
1764        model.add(Conv1D(256, 2, activation='relu',
1765        kernel_initializer='he_uniform',padding='same'))
1766        model.add(MaxPooling1D(pool_size=1))
1767
1768        model.add(Conv1D(384,2, activation='relu',
1769        kernel_initializer='he_uniform',padding='same'))
1770        model.add(Conv1D(384, 2, activation='relu',
1771        kernel_initializer='he_uniform',padding='same'))
1772        model.add(Conv1D(384, 2, activation='relu',
1773        kernel_initializer='he_uniform',padding='same'))
1774        model.add(MaxPooling1D(pool_size=1))
1775
1776
1777        model.add(Flatten())
1778        #malware_model.add(Dropout(0.5))
1779
1780        model.add(Dense(4096, activation='relu',
1781        kernel_initializer='he_uniform'))
1782        model.add(Dropout(0.5))
1783        model.add(Dense(4096, activation='relu',
1784        kernel_initializer='he_uniform'))
1785        model.add(Dropout(0.5))
1786        model.add(Dense(1000, activation='relu',
1787        kernel_initializer='he_uniform'))
1788
1789        model.add(Dense(num_classes, activation='softmax'))
1790
1791        # compile model
1792        opt = SGD(lr=learning_rate, momentum=momentum)
1793        #malware_model.compile(optimizer=opt,
1794        loss='categorical_crossentropy',
1795        metrics=[tf.keras.metrics.Precision(),
1796        tf.keras.metrics.Recall(), 'accuracy'])
1797        model.compile(loss='categorical_crossentropy',
1798        optimizer=opt, metrics=['accuracy'])
1799
1800        return model
1801
1802
1803    def model_Resent(learning_rate=0.01, momentum=0.9):
1804
1805
1806        model = Sequential()
1807
1808        model.add(Conv1D(64, 2, activation="relu",
1809        input_shape=(X.shape[1],1)))
1810        model.add(Dropout(0.5))
1811        model.add(MaxPooling1D(pool_size=1))
1812
1813        model.add(Conv1D(64, 2, activation='relu',
1814        kernel_initializer='he_uniform',padding='same'))
1815        model.add(Conv1D(64, 2, activation='relu',
1816        kernel_initializer='he_uniform',padding='same'))
1817        model.add(Conv1D(64, 2, activation='relu',
1818        kernel_initializer='he_uniform',padding='same'))
1819        model.add(Conv1D(64, 2, activation='relu',
1820        kernel_initializer='he_uniform',padding='same'))
1821
1822        model.add(Conv1D(128,2, activation='relu',
```

```
    kernel_initializer='he_uniform',padding='same'))
    model.add(Dropout(0.5))

    model.add(Conv1D(128, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(128,2, activation='relu',
    kernel_initializer='he_uniform', padding='same'))
    model.add(Conv1D(128, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(128,2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(128, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(128,2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(128, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))

    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Dropout(0.5))

    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform', padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(256, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))

    model.add(Conv1D(512,2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Dropout(0.5))

    model.add(Conv1D(512, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(512, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(512, 2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(512,2, activation='relu',
    kernel_initializer='he_uniform',padding='same'))
    model.add(Conv1D(512, 2, activation='relu',
```

```
1881        kernel_initializer='he_uniform',padding='same'))
1882
1883        model.add(MaxPooling1D(pool_size=1))
1884
1885
1886        model.add(Flatten())
1887        model.add(Dropout(0.5))
1888
1889        model.add(Dense(1000, activation='relu',
1890        kernel_initializer='he_uniform'))
1891
1892        model.add(Dense(num_classes, activation='softmax'))
1893
1894        # compile model
1895        opt = SGD(lr=learning_rate, momentum=momentum)
1896        #malware_model.compile(optimizer=opt,
1897        loss='categorical_crossentropy',
1898        metrics=[tf.keras.metrics.Precision(),
1899        tf.keras.metrics.Recall(), 'accuracy'])
1900        model.compile(loss='categorical_crossentropy',
1901        optimizer=opt, metrics=['accuracy'])
1902
1903        return model
1904
1905    def model_LSTM(learning_rate=0.001, momentum=0.9):
1906
1907        input_layer = Input(shape=(X.shape[1],1))
1908
1909        conv1 = Conv1D(filters=35,
1910                    kernel_size=8,
1911                    strides=1,
1912                    activation='relu')(input_layer)
1913        pool1 = MaxPooling1D(pool_size=4)(conv1)
1914        lstm1 = LSTM(35)(pool1)
1915        output_layer = Dense(7, activation='softmax')(lstm1)
1916        model = Model(inputs=input_layer, outputs=output_layer)
1917        opt = SGD(lr=learning_rate, momentum=momentum)
1918        model.compile(loss='categorical_crossentropy',
1919        optimizer=opt, metrics=['acc'])
1920
1921        return model
1922
1923
1924
1925    print(unique(y_test))
1926    Vuln_model1 = model_VGG16()
1927    Vuln_model2 = model_VGG19()
1928    Vuln_model3 = model_AlexNet()
1929    Vuln_model4 = model_Resent()
1930    Vuln_model5 = model_LSTM()
1931
1932
1933
1934
1935    y_train_new = np.argmax(y_train, axis=1)
1936
1937    y_train_new
1938
1939    from sklearn.utils import class_weight
1940    from sklearn.utils import compute_class_weight
1941
1942    class_weights =
```

```
compute_class_weight(

                            class_weight = "balanced",
                            classes = np.unique(y_train_new),
                            y = y_train_new
                        )
class_weights = dict(zip(np.unique(y_train_new), class_weights))
class_weights



history1 = Vuln_model1.fit(x= Xtrain_scaled,
y=y_train,batch_size=128, epochs=50, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores1 = Vuln_model1.evaluate(x= Xtest_scaled,
y=y_test, verbose=1)
print('Final Vuln_model1 (VGG16) accuracy: ', scores1[1])




plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])

plt.title('VGG16 model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])

plt.title('VGG16 model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()


classes = Vuln_model1.predict(x= Xtest_scaled)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)

print(classification_report(true_classes, y_classes,digits=6))

cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

history2 = Vuln_model2.fit(x= Xtrain_scaled, y=y_train,
batch_size=128, epochs=20, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores2 = Vuln_model2.evaluate(x= Xtest_scaled,
y=y_test, verbose=1)
print('Final Vuln_model2 (VGG19) accuracy: ', scores1[1])

import matplotlib.pyplot as plt2


```

```
plt2.plot(history2.history['accuracy'])
plt2.plot(history2.history['val_accuracy'])

plt2.title('VGG19 model accuracy')
plt2.ylabel('accuracy')
plt2.xlabel('epoch')
plt2.legend(['train','test'], loc='upper left')
plt2.show()

plt2.plot(history2.history['loss'])
plt2.plot(history2.history['val_loss'])

plt2.title('VGG19 model loss')
plt2.ylabel('loss')
plt2.xlabel('epoch')
plt2.legend(['train','test'], loc='upper left')
plt2.show()


classes = Vuln_model2.predict(x= Xtest_scaled)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)

print(classification_report(true_classes, y_classes,digits=6))


cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

import matplotlib.pyplot as plt3


history3 = Vuln_model3.fit(x= Xtrain_scaled, y=y_train,
batch_size=128, epochs=20, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores3 = Vuln_model3.evaluate(x= Xtest_scaled,
y=y_test, verbose=1)
print('Final Vuln_model3 (AlexNet) accuracy: ', scores3[1])



plt3.plot(history3.history['accuracy'])
plt3.plot(history3.history['val_accuracy'])

plt3.title('AlexNet model accuracy')
plt3.ylabel('accuracy')
plt3.xlabel('epoch')
plt3.legend(['train','test'], loc='upper left')
plt3.show()

plt3.plot(history3.history['loss'])
plt3.plot(history3.history['val_loss'])

plt3.title('AlexNet model loss')
plt3.ylabel('loss')
plt3.xlabel('epoch')
plt3.legend(['train','test'], loc='upper left')
plt3.show()
```

```
classes = Vuln_model3.predict(x= Xtest_scaled)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)
print(classification_report(true_classes, y_classes,digits=6))



cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

import matplotlib.pyplot as plt4


history4 = Vuln_model4.fit(x= Xtrain_scaled,
y=y_train, batch_size=128, epochs=20, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores4 = Vuln_model4.evaluate(x= Xtest_scaled,
y=y_test, verbose=1)
print('Final Vuln_model4 (Resent) accuracy: ', scores4[1])


plt4.plot(history4.history['accuracy'])
plt4.plot(history4.history['val_accuracy'])

plt4.title('Resent model accuracy')
plt4.ylabel('accuracy')
plt4.xlabel('epoch')
plt4.legend(['train','test'], loc='upper left')
plt4.show()

plt4.plot(history4.history['loss'])
plt4.plot(history4.history['val_loss'])

plt4.title('Resent model loss')
plt4.ylabel('loss')
plt4.xlabel('epoch')
plt4.legend(['train','test'], loc='upper left')
plt4.show()


classes = Vuln_model4.predict(x= Xtest_scaled)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)
print(classification_report(true_classes, y_classes,digits=6))



cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

history5 = Vuln_model5.fit(x= X_train, y=y_train,
batch_size=128, epochs=20, verbose=1,
validation_split=0.33,class_weight=class_weights)
scores5 = Vuln_model5.evaluate(x= X_test,
y=y_test, verbose=1)
print('Final Vuln_model5 (LSTM) accuracy: ', scores5[1])

import matplotlib.pyplot as plt5

plt5.plot(history5.history['acc'])
```

```
plt5.plot(history5.history['val_acc'])

plt5.title('LSTM model accuracy')
plt5.ylabel('accuracy')
plt5.xlabel('epoch')
plt5.legend(['train','test'], loc='upper left')
plt5.show()

plt5.plot(history5.history['loss'])
plt5.plot(history5.history['val_loss'])

plt5.title('LSTM model loss')
plt5.ylabel('loss')
plt5.xlabel('epoch')
plt5.legend(['train','test'], loc='upper left')
plt5.show()


classes = Vuln_model5.predict(x= X_test)
y_classes = classes.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)

print(classification_report(true_classes, y_classes,digits=6))



cf_matrix = confusion_matrix(true_classes, y_classes)
print(cf_matrix)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2\%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')




Vuln_model1.save('/Users/abdullah/models/Model1.h5')
Vuln_model2.save('/Users/abdullah/models/Model2.h5')
Vuln_model3.save('/Users/abdullah/models/Model3.h5')
Vuln_model4.save('/Users/abdullah/models/Model4.h5')
Vuln_model5.save('/Users/abdullah/models/Model5.h5')


from keras.models import load_model

# load models from file
def load_all_models(n_models):
        all_models = list()
        for i in range(n_models):
                # define filename for this ensemble
                filename = '/Users/abdullah/models/Model' + str(i+1) + '.h5'
                # load model from file
                model = load_model(filename)
                # add to list of members
```

```
                all_models.append(model)
                print('>loaded \%s' \% filename)
        return all_models


# load all models
n_members = 5
members = load_all_models(n_members)
print('Loaded \%d models' \% len(members))


 # create stacked model input dataset as outputs from the ensemble
def stacked_dataset(members, inputX):
        stackX = None
        for model in members:
                # make prediction
                yhat = model.predict(inputX, verbose=0)
                # stack predictions into [rows, members, probabilities]
                if stackX is None:
                        stackX = yhat
                else:
                        stackX = dstack((stackX, yhat))
        # flatten predictions to [rows, members x probabilities]
        stackX = stackX.reshape((stackX.shape[0],
 stackX.shape[1]*stackX.shape[2]))
        return stackX




 # fit a model based on the outputs from the ensemble members
def fit_stacked_model(members, inputX, inputy):
        # create dataset using ensemble
        stackedX = stacked_dataset(members, inputX)
        # fit standalone model
        model = LogisticRegression()
        model.fit(stackedX, inputy)
        return model


# fit stacked model using the ensemble
from numpy import dstack
from sklearn.linear_model import LogisticRegression
y_test=np.argmax(y_test, axis=1)

model = fit_stacked_model(members, Xtest_scaled, y_test)



# make a prediction with the stacked model

def stacked_prediction(members, model, inputX):
        # create dataset using ensemble
        stackedX = stacked_dataset(members, inputX)
        # make a prediction
        yhat = model.predict(stackedX)
        return yhat

# evaluate model on test set
from sklearn.metrics import accuracy_score
yhat = stacked_prediction(members, model, Xtest_scaled)

acc = accuracy_score(y_test, yhat)
```

```
print('Stacked Test Accuracy: \%.3f' \% acc)
print(len(yhat))


'''
y_classes = yhat.argmax(axis=-1)
true_classes = y_test.argmax(axis=-1)

print(yhat)
print(y_classes)
print(y_test)
print(true_classes)

for x in range(len(y_classes)):
    print(classes[x])
print("++++++")
print(true_classes)
for x in range(len(true_classes)):
    print(true_classes[x])
'''
print(classification_report(y_test, yhat, digits=6))

cf_matrix = confusion_matrix(y_test, yhat)
print(cf_matrix)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2\%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

REFERENCES

Ferenc, R., Hegedűs, P., Gyimesi, P., Antal, G., Bán, D., and Gyimóthy, T. (2019). Challenging machine learning algorithms in predicting vulnerable javascript functions. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 8–14. IEEE.

Ganesh, S., Ohlsson, T., and Palma, F. (2021). Predicting security vulnerabilities using source code metrics. In *2021 Swedish Workshop on Data Science (SweDS)*, pages 1–7. IEEE.

Ganesh, S., Palma, F., and Olsson, T. (2022). Are source code metrics "good enough" in predicting security vulnerabilities? *Data*, 7(9):127.

Viszkok, T., Hegedűs, P., and Ferenc, R. (2021). Improving vulnerability prediction of javascript functions using process metrics. *arXiv preprint arXiv:2105.07527*.