

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

Abdullah Sheneamer¹

¹Department of Computer Science, Jazan University, Jazan, KSA

Corresponding author:

Abdullah Sheneamer¹

Email address: asheneamer@jazanu.edu.sa

ABSTRACT

Keywords:

DATA

- Ferenc et al's Dataset 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
```

```

43 from sklearn.model_selection import train_test_split
44 from sklearn.metrics import classification_report, accuracy_score,
45 confusion_matrix
46 import pickle
47 # loading library
48 import numpy as np # linear algebra
49 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
50 from sklearn.model_selection import train_test_split
51 from sklearn.neighbors import KNeighborsClassifier
52 from sklearn.ensemble import RandomForestClassifier
53 from sklearn.ensemble import BaggingClassifier
54 from sklearn.tree import DecisionTreeClassifier
55 from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
56 from sklearn import svm
57 from imread import imread, imsave
58 from PIL import Image
59 import numpy as np # linear algebra
60 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
61 from sklearn.model_selection import train_test_split
62 from sklearn.preprocessing import MinMaxScaler
63 from keras.models import Model
64 from keras.layers import Input
65 import seaborn as sns
66 from keras.layers.core import Activation, Dropout, Dense
67 from sklearn import preprocessing
68 from sklearn.metrics import confusion_matrix
69 from sklearn.metrics import classification_report
70 import numpy as np # linear algebra
71 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
72 from sklearn.model_selection import train_test_split
73 import sys
74 import os
75 from math import log
76 import scipy as sp
77 from imblearn.over_sampling import RandomOverSampler, SMOTE
78 from imblearn.under_sampling import RandomUnderSampler
79
80 import matplotlib.pyplot as plt
81 from keras.optimizers import SGD
82 import tensorflow as tf
83 from keras.models import Sequential
84 from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
85 from sklearn.model_selection import train_test_split
86 from sklearn.datasets import load_iris
87 from numpy import unique
88 from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN, Embedding,
89 Reshape
90 from sklearn.metrics import confusion_matrix
91 from collections import Counter
92 from xgboost import XGBClassifier
93
94
95 df = pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
96 JSVulnerabilityDataSet-1.0.csv')
97 #df = pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
98 MDPIData2022-main/datasets/tomcat-final.csv')
99 #df = pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
100 MDPIData2022-main/datasets/struts-final.csv')
101
102
103
104 import matplotlib.pyplot as plt
105 df.columns
106
107

```

```

108 sns.countplot(x='Vuln', data=df)
109
110 X=df[[['CC', 'CCL', 'CCO', 'CI', 'CLC', 'CLLC', 'MCC',
111        'NL', 'NLE', 'CD', 'CLOC', 'DLOC', 'TCD', 'TCLOC',
112        'LLOC', 'LOC', 'NOS', 'NUMPAR', 'TLLOC', 'TLOC',
113        'TNOS', 'HOR_D', 'HOR_T', 'HON_D', 'HON_T',
114        'HLEN', 'HVOC', 'HDIFF', 'HVOL', 'HEFF',
115        'HBUGS', 'HTIME', 'CYCL', 'PARAMS',
116        'CYCL_DENS']] . values
117
118
119
120 #y=df['Vuln']
121 y=df['Vuln']
122 #label_encoder object knows how to understand word labels .
123 label_encoder=preprocessing.LabelEncoder()
124
125 #Encode labels in column 'species' .
126 y=label_encoder.fit_transform(y)
127
128 counter=Counter(y)
129 print(counter)
130
131 #ros=RandomOverSampler(sampling_strategy={0:10629,1:10629},
132 random_state=42)#String
133 #X, y=ros.fit_resample(X, y)
134
135 #rus=RandomUnderSampler(sampling_strategy={0:1496,1:1496},
136 random_state=42)#String
137 #X, y=rus.fit_resample(X, y)
138
139 #smote=SMOTE()#SMOTE(" minority ")
140 #X, y=smote.fit_resample(X, y)
141
142
143
144 counter=Counter(y)
145 print(counter)
146
147
148 X_train, X_test, y_train, y_test=train_test_split(X, y,
149 test_size=0.33, shuffle=True, random_state=42, stratify=y)
150 X_test1=X_test
151 X_train1=X_train
152 y_test1=y_test
153 y_train1=y_train
154
155
156 from tensorflow.keras.utils import to_categorical
157 y_train=to_categorical(y_train)
158 y_test=to_categorical(y_test)
159
160
161
162 from sklearn.preprocessing import MinMaxScaler
163 scaler=MinMaxScaler()
164 scaler.fit(X_train)
165
166 Xtrain_scaled=scaler.transform(X_train)
167 Xtest_scaled=scaler.transform(X_test)
168
169 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 ")

```

```

231     scores = cross_val_score(model, X, y,
232     scoring=make_scorer(classification_report_with_accuracy_score)
233     , cv=cv, n_jobs=-1, error_score='raise')
234     scores = cross_val_score(model, X, y, scoring="f1", cv=cv,
235     n_jobs=-1, error_score='raise')
236     return scores
237
238
239 # get the models to evaluate
240 models = get_models()
241 # evaluate the models and store results
242 results, names = list(), list()
243 for name, model in models.items():
244     scores = evaluate_model(model, X, y, name)
245     results.append(scores)
246     names.append(name)
247     print("====="+name+"=====")
248     print("=====")
249     #print(name)
250     cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
251     random_state=42)
252     cv_s1 = cross_val_score(model, X, y, scoring=
253     make_scorer(classification_report_with_accuracy_score),
254     cv=cv, n_jobs=-1, error_score='raise').mean()
255     for score in ["roc_auc", "f1", "precision", "recall", "accuracy"]:
256         cv_s = cross_val_score(model, X, y, scoring=score, cv=cv,
257         n_jobs=-1, error_score='raise').mean()
258         print(score+" : "+str(cv_s))
259     #print(cv_s1)
260     print('\n')
261     pyplot.boxplot(results, labels=names, showmeans=True)
262     pyplot.show()

```

263 0.2 Deep learning for Binary classes

```

264 import numpy as np # linear algebra
265 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
266 from sklearn.model_selection import train_test_split
267 from sklearn.preprocessing import MinMaxScaler
268 from keras.models import Model
269 from keras.layers import Input
270 import seaborn as sns
271 from keras.layers.core import Activation, Dropout, Dense
272 from sklearn import preprocessing
273 from sklearn.metrics import confusion_matrix
274 from sklearn.metrics import classification_report
275 import numpy as np # linear algebra
276 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
277 from sklearn.model_selection import train_test_split
278 import sys
279 import os
280 from math import log
281 import scipy as sp
282 from imblearn.over_sampling import RandomOverSampler, SMOTE
283 from imblearn.under_sampling import RandomUnderSampler
284
285 import matplotlib.pyplot as plt
286 from keras.optimizers import SGD
287 import tensorflow as tf
288 from keras.models import Sequential
289 from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
290 from sklearn.model_selection import train_test_split

```

```

291 from sklearn.datasets import load_iris
292 from numpy import unique
293 from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN,
294 Embedding, Reshape
295 from sklearn.metrics import confusion_matrix
296 from collections import Counter
297
298 df = pd.read_csv('/Users/abdullah/Desktop/Folder/Research/
299 JSVulnerabilityDataSet-1.0.csv')
300 import matplotlib.pyplot as plt
301 sns.countplot(x='Vuln', data=df)
302
303 X = df[['CC', 'CCL', 'CCO', 'CI', 'CLC', 'CLLC', 'MeCC',
304         'NL', 'NLE', 'CD', 'CLOC', 'DLOC', 'TCD', 'TCLOC',
305         'LLOC', 'LOC', 'NOS', 'NUMPAR', 'TLLOC', 'TLOC',
306         'TNOS', 'HOR_D', 'HOR_T', 'HON_D', 'HON_T', 'HLEN',
307         'HVOC', 'HDIFF', 'HVOL', 'HEFF', 'HBUGS', 'HTIME',
308         'CYCL', 'PARAMS', 'CYCL_DENS']].values
309
310 y = df['Vuln']
311
312 y
313
314 # label_encoder object knows how to understand word labels.
315 label_encoder = preprocessing.LabelEncoder()
316
317 # Encode labels in column 'species'.
318 y = label_encoder.fit_transform(y)
319
320 counter = Counter(y)
321 print(counter)
322
323 #ros = RandomOverSampler(sampling_strategy={ 0: 10629,1: 10629},
324 random_state=42) # String
325 #X, y = ros.fit_resample(X, y)
326
327 #rus = RandomUnderSampler(sampling_strategy={ 0: 1496,1: 1496},
328 random_state=42) # String
329 #X, y = rus.fit_resample(X, y)
330
331 #smote = SMOTE() #SMOTE("minority")
332 #X, y = smote.fit_resample(X, y)
333
334
335
336 counter = Counter(y)
337 print(counter)
338
339
340 X_train, X_test, y_train, y_test = train_test_split(X,y,
341 test_size=0.33,shuffle=True, random_state=42, stratify=y)
342 X_test1=X_test
343 X_train1=X_train
344 y_test1=y_test
345 y_train1=y_train
346
347
348 from tensorflow.keras.utils import to_categorical
349 y_train = to_categorical(y_train)
350 y_test = to_categorical(y_test)
351
352
353

```

```

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.compile(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

```

```

653     return model
654
655 def model_LSTM(learning_rate=0.001, momentum=0.9):
656
657     input_layer = Input(shape=(X.shape[1],1))
658
659     conv1 = Conv1D(filters=35,
660                   kernel_size=8,
661                   strides=1,
662                   activation='relu')(input_layer)
663     pool1 = MaxPooling1D(pool_size=4)(conv1)
664     lstm1 = LSTM(35)(pool1)
665     output_layer = Dense(2, activation='sigmoid')(lstm1)
666     model = Model(inputs=input_layer, outputs=output_layer)
667     opt = SGD(lr=learning_rate, momentum=momentum)
668     model.compile(loss='binary_crossentropy',
669                 optimizer=opt, metrics=['acc'])
670
671     return model
672
673
674
675 print(unique(y_test))
676 Vuln_model1 = model_VGG16()
677 Vuln_model2 = model_VGG19()
678 Vuln_model3 = model_AlexNet()
679 Vuln_model4 = model_Resnet()
680 Vuln_model5 = model_LSTM()
681
682
683
684
685 y_train_new = np.argmax(y_train, axis=1)
686
687 y_train_new
688
689 from sklearn.utils import class_weight
690 from sklearn.utils import compute_class_weight
691
692 class_weights = compute_class_weight(
693                                     class_weight = "balanced",
694                                     classes = np.unique(y_train_new),
695                                     y = y_train_new
696                                 )
697 class_weights = dict(zip(np.unique(y_train_new), class_weights))
698 class_weights
699
700
701
702
703 history1 = Vuln_model1.fit(x= Xtrain_scaled,
704 y=y_train, batch_size=128, epochs=1000, verbose=1,
705 validation_split=0.33, class_weight=class_weights)
706 scores1 = Vuln_model1.evaluate(x= Xtest_scaled,
707 y=y_test, verbose=1)
708 print('Final Vuln_model1(VGG16) accuracy:', scores1[1])
709
710 plt.plot(history1.history['accuracy'])
711 plt.plot(history1.history['val_accuracy'])
712
713 plt.title('VGG16_model_accuracy')
714 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:', scores2[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

```

```

836 cf_matrix = confusion_matrix(true_classes, y_classes)
837 print(cf_matrix)
838
839 import matplotlib.pyplot as plt4
840
841 history4 = Vuln_model4.fit(x= Xtrain_scaled,
842 y=y_train, batch_size=128, epochs=1000, verbose=1,
843 validation_split=0.33, class_weight=class_weights)
844 scores4 = Vuln_model4.evaluate(x= Xtest_scaled,
845 y=y_test, verbose=1)
846 print('Final Vuln_model4 (Resent) accuracy:', scores4[1])
847
848
849 print("Test Score:", scores4[0])
850 print("Test Accuracy:", scores4[1])
851
852 plt4.plot(history4.history['accuracy'])
853 plt4.plot(history4.history['val_accuracy'])
854
855 plt4.title('Resent_model_accuracy')
856 plt4.ylabel('accuracy')
857 plt4.xlabel('epoch')
858 plt4.legend(['train', 'test'], loc='upper_left')
859 plt4.show()
860
861 plt4.plot(history4.history['loss'])
862 plt4.plot(history4.history['val_loss'])
863
864 plt4.title('Resent_model_loss')
865 plt4.ylabel('loss')
866 plt4.xlabel('epoch')
867 plt4.legend(['train', 'test'], loc='upper_left')
868 plt4.show()
869
870
871 classes = Vuln_model4.predict(x= Xtest_scaled)
872 y_classes = classes.argmax(axis=-1)
873 true_classes = y_test.argmax(axis=-1)
874 print(classification_report(true_classes, y_classes, digits=6))
875
876 cf_matrix = confusion_matrix(true_classes, y_classes)
877 print(cf_matrix)
878
879 history5 = Vuln_model5.fit(x= X_train, y=y_train,
880 batch_size=128, epochs=1000, verbose=1,
881 validation_split=0.33, class_weight=class_weights)
882 scores5 = Vuln_model5.evaluate(x= X_test,
883 y=y_test, verbose=1)
884 print('Final Vuln_model5 (LSTM) accuracy:', scores5[1])
885
886 import matplotlib.pyplot as plt5
887 print("Test Score:", scores5[0])
888 print("Test Accuracy:", scores5[1])
889
890 plt5.plot(history5.history['acc'])
891 plt5.plot(history5.history['val_acc'])
892
893 plt5.title('LSTM_model_accuracy')
894 plt5.ylabel('accuracy')
895 plt5.xlabel('epoch')

```

```

896 plt5.legend(['train', 'test'], loc='upper_left')
897 plt5.show()
898
899 plt5.plot(history5.history['loss'])
900 plt5.plot(history5.history['val_loss'])
901
902 plt5.title('LSTM_model_loss')
903 plt5.ylabel('loss')
904 plt5.xlabel('epoch')
905 plt5.legend(['train', 'test'], loc='upper_left')
906 plt5.show()
907
908
909 classes = Vuln_model5.predict(x= X_test)
910 y_classes = classes.argmax(axis=-1)
911 true_classes = y_test.argmax(axis=-1)
912
913 print(classification_report(true_classes, y_classes, digits=6))
914
915
916
917 cf_matrix = confusion_matrix(true_classes, y_classes)
918 print(cf_matrix)
919
920 group_names = ['TrueNeg', 'FalsePos', 'FalseNeg', 'TruePos']
921 group_counts = ["{0:0.0f}".format(value) for value in
922                 cf_matrix.flatten()]
923 group_percentages = ["{0:.2%}".format(value) for value in
924                      cf_matrix.flatten()/np.sum(cf_matrix)]
925 labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
926           zip(group_names, group_counts, group_percentages)]
927 labels = np.asarray(labels).reshape(2,2)
928 sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
929
930
931
932
933 Vuln_model1.save('/Users/abdullah/models/Model1.h5')
934 Vuln_model2.save('/Users/abdullah/models/Model2.h5')
935 Vuln_model3.save('/Users/abdullah/models/Model3.h5')
936 Vuln_model4.save('/Users/abdullah/models/Model4.h5')
937 Vuln_model5.save('/Users/abdullah/models/Model5.h5')
938
939
940
941 from keras.models import load_model
942
943 # load models from file
944 def load_all_models(n_models):
945     all_models = list()
946     for i in range(n_models):
947         # define filename for this ensemble
948         filename = '/Users/abdullah/models/Model' + str(i+1) + '.h5'
949         # load model from file
950         model = load_model(filename)
951         # add to list of members
952         all_models.append(model)
953         print('>loaded_\%s' \% filename)
954     return all_models
955
956

```



```

957 # load all models
958 n_members = 5
959 members = load_all_models(n_members)
960 print('Loaded %d models' % len(members))
961
962
963 # create stacked model input dataset as outputs from the ensemble
964 def stacked_dataset(members, inputX):
965     stackX = None
966     for model in members:
967         # make prediction
968         yhat = model.predict(inputX, verbose=0)
969         # stack predictions into [rows, members, probabilities]
970         if stackX is None:
971             stackX = yhat
972         else:
973             stackX = dstack((stackX, yhat))
974     # flatten predictions to [rows, members x probabilities]
975     stackX = stackX.reshape((stackX.shape[0],
976                             stackX.shape[1]*stackX.shape[2]))
977     return stackX
978
979
980
981
982 # fit a model based on the outputs from the ensemble members
983 def fit_stacked_model(members, inputX, inputy):
984     # create dataset using ensemble
985     stackedX = stacked_dataset(members, inputX)
986     # fit standalone model
987     model = LogisticRegression()
988     model.fit(stackedX, inputy)
989     return model
990
991
992 # fit stacked model using the ensemble
993 from numpy import dstack
994 from sklearn.linear_model import LogisticRegression
995 y_test=np.argmax(y_test, axis=1)
996
997 model = fit_stacked_model(members, Xtest_scaled, y_test)
998
999
1000
1001 # make a prediction with the stacked model
1002
1003 def stacked_prediction(members, model, inputX):
1004     # create dataset using ensemble
1005     stackedX = stacked_dataset(members, inputX)
1006     # make a prediction
1007     yhat = model.predict(stackedX)
1008     return yhat
1009
1010 # evaluate model on test set
1011 from sklearn.metrics import accuracy_score
1012 yhat = stacked_prediction(members, model, Xtest_scaled)
1013
1014 acc = accuracy_score(y_test, yhat)
1015 print('Stacked Test Accuracy: %.3f' % acc)
1016 print(len(yhat))
1017
1018
1019 print(classification_report(y_test, yhat, digits=6))

```

```

1020
1021 cf_matrix = confusion_matrix(y_test, yhat)
1022 print(cf_matrix)
1023
1024 group_names = ['True□Neg', 'False□Pos', 'False□Neg', 'True□Pos']
1025 group_counts = [{"0:0.0f"}.format(value) for value in
1026                 cf_matrix.flatten()]
1027 group_percentages = [{"0:.2\%"}.format(value) for value in
1028                      cf_matrix.flatten()/np.sum(cf_matrix)]
1029 labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
1030           zip(group_names, group_counts, group_percentages)]
1031 labels = np.asarray(labels).reshape(2,2)
1032 sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
1033 }

```

1034 0.3 Machine learning for Multi-classes

```

1035 # compare ensemble to each baseline classifier
1036 from numpy import mean
1037 from numpy import std
1038 from sklearn.datasets import make_classification
1039 from sklearn.model_selection import cross_val_score
1040 from sklearn.model_selection import RepeatedStratifiedKFold
1041 from sklearn.linear_model import LogisticRegression
1042 from sklearn.neighbors import KNeighborsClassifier
1043 from sklearn.tree import DecisionTreeClassifier
1044 from sklearn.svm import SVC
1045 from sklearn.naive_bayes import GaussianNB
1046 from sklearn.ensemble import StackingClassifier
1047 from matplotlib import pyplot
1048 import pandas as pd
1049 from sklearn import svm
1050 from sklearn.model_selection import GridSearchCV
1051 import os
1052 import matplotlib.pyplot as plt
1053 #from skimage.transform import resize
1054 import imread
1055 import numpy as np
1056 from sklearn.model_selection import train_test_split
1057 from sklearn.metrics import
1058 classification_report, accuracy_score, confusion_matrix
1059 import pickle
1060 # loading library
1061 import numpy as np # linear algebra
1062 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
1063 from sklearn.model_selection import train_test_split
1064 from sklearn.neighbors import KNeighborsClassifier
1065 from sklearn.ensemble import RandomForestClassifier
1066 from sklearn.ensemble import BaggingClassifier
1067 from sklearn.tree import DecisionTreeClassifier
1068 from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
1069 from sklearn import svm
1070 from imread import imread, imsave
1071 from PIL import Image
1072 import numpy as np # linear algebra
1073 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
1074 from sklearn.model_selection import train_test_split
1075 from sklearn.preprocessing import MinMaxScaler
1076 from keras.models import Model
1077 from keras.layers import Input
1078 import seaborn as sns
1079 from keras.layers.core import Activation, Dropout, Dense
1080 from sklearn import preprocessing
1081 from sklearn.metrics import confusion_matrix

```

```

1082 from sklearn.metrics import classification_report
1083 import numpy as np # linear algebra
1084 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
1085 from sklearn.model_selection import train_test_split
1086 import sys
1087 import os
1088 from math import log
1089 import scipy as sp
1090 from imblearn.over_sampling import RandomOverSampler, SMOTE
1091 from imblearn.under_sampling import RandomUnderSampler
1092
1093 import matplotlib.pyplot as plt
1094 from keras.optimizers import SGD
1095 import tensorflow as tf
1096 from keras.models import Sequential
1097 from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
1098 from sklearn.model_selection import train_test_split
1099 from sklearn.datasets import load_iris
1100 from numpy import unique
1101 from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN, Embedding, Res
1102 from sklearn.metrics import confusion_matrix
1103 from collections import Counter
1104 from xgboost import XGBClassifier
1105 import matplotlib.pyplot as plt
1106
1107 df =pd.read_csv( '/Users/abdullah/Desktop/Folder/
1108 Research/Vulnerability_Research/Vul_Datasets/
1109 Code_Metrics_Datasets/Multi-Classes/PU_Dataset.csv ')
1110 df1 =pd.read_csv( '/Users/abdullah/Desktop/Folder/
1111 Research/Vulnerability_Research/Vul_Datasets/
1112 Code_Metrics_Datasets/Multi-Classes/cwe119_GCDFFile.csv ')
1113 df2 =pd.read_csv( '/Users/abdullah/Desktop/Folder/Research/
1114 Vulnerability_Research/Vul_Datasets/
1115 Code_Metrics_Datasets/Multi-Classes/cwe399_cgd.csv ')
1116 df3 =pd.read_csv( '/Users/abdullah/Desktop/Folder
1117 /Research/Vulnerability_Research/Vul_Datasets/
1118 Code_Metrics_Datasets/Multi-Classes/API_function_call.csv ')
1119 df4 =pd.read_csv( '/Users/abdullah/Desktop/Folder
1120 /Research/Vulnerability_Research/Vul_Datasets/
1121 Code_Metrics_Datasets/Multi-Classes/Array_usage.csv ')
1122 df5 =pd.read_csv( '/Users/abdullah/Desktop/Folder/
1123 Research/Vulnerability_Research/Vul_Datasets/
1124 Code_Metrics_Datasets/Multi-Classes/Arithmetic_expression.csv ')
1125
1126 import matplotlib.pyplot as plt
1127 df.columns
1128
1129
1130 sns.countplot(x='IsVulnerable', data=df)
1131
1132 X = df[[ 'Lines_Of_Program', 'Physic_Lines',
1133         'n1_Number_Of_Distinct_Operators',
1134         'n2_Number_Of_Distinct_Operands', 'n_Program_Vocabulary',
1135         'N1_Total_Number_Of_Operators', 'N2_Total_Number_Of_Operands',
1136         'N_Program_Length', 'B_Number_of_Delivered_Bugs_1',
1137         'B_Number_of_Delivered_Bugs_2',
1138         'D_Difficulty', 'E_Effort', 'T_Time_Required_To_Program',
1139         'V_Volume', '_N_Calculated_Program_Length',
1140         'McCab_Number' ]].values
1141
1142
1143 y= df[ 'IsVulnerable' ].values
1144

```

```

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

```

```

1209 X4 = df4[['Lines_Of_Program', 'Physic_Lines',
1210          'n1_Number_Of_Distinct_Operators',
1211          'n2_Number_Of_Distinct_Operands', 'n_Program_Vocabulary',
1212          'N1_Total_Number_Of_Operators', 'N2_Total_Number_Of_Operands',
1213          'N_Program_Length', 'B_Number_of_Delivered_Bugs_1',
1214          'B_Number_of_Delivered_Bugs_2', 'D_Difficulty',
1215          'E_Effort', 'T_Time_Required_To_Program', 'V_Volume',
1216          '_N_Calculated_Program_Length', 'McCab_Number']].values
1217
1218
1219 y4= df4['IsVulnerable'].values
1220
1221 X5 = df5[['Lines_Of_Program', 'Physic_Lines',
1222          'n1_Number_Of_Distinct_Operators',
1223          'n2_Number_Of_Distinct_Operands',
1224          'n_Program_Vocabulary', 'N1_Total_Number_Of_Operators',
1225          'N2_Total_Number_Of_Operands', 'N_Program_Length',
1226          'B_Number_of_Delivered_Bugs_1', 'B_Number_of_Delivered_Bugs_2',
1227          'D_Difficulty', 'E_Effort', 'T_Time_Required_To_Program',
1228          'V_Volume', '_N_Calculated_Program_Length',
1229          'McCab_Number']].values
1230
1231
1232 y5= df5['IsVulnerable'].values
1233
1234
1235 print(len(y5))
1236
1237
1238 import numpy as np
1239 X=np.concatenate((X,X1,X2,X3,X4,X5))
1240 y=np.concatenate((y,y1,y2,y3,y4,y5))
1241
1242 import numpy
1243 print(numpy.unique(y))
1244
1245 # label_encoder object knows how to understand word labels.
1246 label_encoder = preprocessing.LabelEncoder()
1247
1248 # Encode labels in column 'species'.
1249 y = label_encoder.fit_transform(y)
1250 print(numpy.unique(y))
1251
1252 #from sklearn.preprocessing import label_binarize
1253 #y = label_binarize(y, classes=[0, 1, 2, 3,4,5,6])
1254
1255 #counter = Counter(y)
1256 #print(counter)
1257
1258 X_train, X_test, y_train, y_test = train_test_split(X,y,
1259 test_size=0.33,shuffle=True, random_state=42, stratify=y)
1260 X_test1=X_test
1261 X_train1=X_train
1262 y_test1=y_test
1263 y_train1=y_train
1264
1265
1266
1267
1268
1269 from tensorflow.keras.utils import to_categorical
1270 y_train = to_categorical(y_train)
1271 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

```

```

1333
1334 from sklearn.metrics import classification_report ,
1335 accuracy_score , make_scorer
1336
1337 # evaluate a give model using cross-validation
1338 def evaluate_model(model, X, y,name):
1339     cv = RepeatedStratifiedKFold(n_splits=10,
1340     n_repeats=3, random_state=1)
1341     print( "===== "+name+"====="
1342     "░░░░░░=====")
1343     #scores = cross_val_score(model, X, y, scoring=make_scorer(
1344     classification_report_with_accuracy_score),
1345     cv=cv, n_jobs=-1, error_score='raise')
1346     scores = cross_val_score(model, X, y, scoring="f1",
1347     cv=cv, n_jobs=-1, error_score='raise')
1348     return scores
1349
1350
1351 # get the models to evaluate
1352 models = get_models()
1353 # evaluate the models and store results
1354 results , names = list() , list()
1355 for name, model in models.items():
1356     #scores = evaluate_model(model, X, y,name)
1357     #results.append(scores)
1358     #names.append(name)
1359     print( "===== "+name+"====="
1360     "░░░░░░░░░░░░=====")
1361     #print(name)
1362     cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
1363     random_state=42)
1364     for score in ["roc_auc", "f1", "precision",
1365     "recall", "accuracy"]:
1366         cvs = cross_val_score(model, X, y,
1367         scoring=score, cv=cv, n_jobs=-1,
1368         error_score='raise').mean()
1369         print(score + " : " + str(cvs))
1370     #print(cvs1)
1371     print(' \n')
1372     pyplot.boxplot(results , labels=names, showmeans=True)
1373     pyplot.show()

```

1374 0.4 Deep learning for Multi-classes

```

1375 import numpy as np # linear algebra
1376 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
1377 from sklearn.model_selection import train_test_split
1378 from sklearn.preprocessing import MinMaxScaler
1379 from keras.models import Model
1380 from keras.layers import Input
1381 import seaborn as sns
1382 from keras.layers.core import Activation , Dropout , Dense
1383 from sklearn import preprocessing
1384 from sklearn.metrics import confusion_matrix
1385 from sklearn.metrics import classification_report
1386 import numpy as np # linear algebra
1387 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
1388 from sklearn.model_selection import train_test_split
1389 import sys
1390 import os
1391 from math import log

```

```

1392 import scipy as sp
1393 from imblearn.over_sampling import RandomOverSampler, SMOTE
1394 from imblearn.under_sampling import RandomUnderSampler
1395
1396 import matplotlib.pyplot as plt
1397 from keras.optimizers import SGD
1398 import tensorflow as tf
1399 from keras.models import Sequential
1400 from keras.layers import Dropout, Dense, Conv1D, Flatten, MaxPooling1D
1401 from sklearn.model_selection import train_test_split
1402 from sklearn.datasets import load_iris
1403 from numpy import unique
1404 from keras.layers import Dense, Input, LSTM, Dropout, SimpleRNN,
1405 Embedding, Reshape
1406 from sklearn.metrics import confusion_matrix
1407 from collections import Counter
1408
1409 df = pd.read_csv('/Users/abdullah/Desktop/Folder/
1410 Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
1411 Multi-Classes/PU_Dataset.csv')
1412 df1 = pd.read_csv('/Users/abdullah/Desktop/Folder/
1413 Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
1414 Multi-Classes/cwe119_GCDFile.csv')
1415 df2 = pd.read_csv('/Users/abdullah/Desktop/Folder/
1416 Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
1417 Multi-Classes/cwe399_cgd.csv')
1418 df3 = pd.read_csv('/Users/abdullah/Desktop/Folder/
1419 Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
1420 Multi-Classes/API_function_call.csv')
1421 df4 = pd.read_csv('/Users/abdullah/Desktop/Folder/
1422 Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
1423 Multi-Classes/Array_usage.csv')
1424 df5 = pd.read_csv('/Users/abdullah/Desktop/Folder/
1425 Research/Vulnerability_Research/Vul_Datasets/Code_Metrics_Datasets/
1426 Multi-Classes/Arithmetic_expression.csv')
1427
1428
1429
1430
1431
1432
1433 import matplotlib.pyplot as plt
1434 df.columns
1435
1436
1437 sns.countplot(x='IsVulnerable', data=df)
1438
1439 X = df[['Lines_Of_Program', 'Physic_Lines',
1440         'n1_Number_Of_Distinct_Operators',
1441         'n2_Number_Of_Distinct_Operands', 'n_Program_Vocabulary',
1442         'N1_Total_Number_Of_Operators', 'N2_Total_Number_Of_Operands',
1443         'N_Program_Length', 'B_Number_of_Delivered_Bugs_1',
1444         'B_Number_of_Delivered_Bugs_2',
1445         'D_Difficulty', 'E_Effort', 'T_Time_Required_To_Program',
1446         'V_Volume', '_N_Calculated_Program_Length',
1447         'McCab_Number']].values
1448
1449
1450 y= df['IsVulnerable'].values
1451
1452
1453
1454 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))

```

```

1519
1520
1521
1522
1523
1524 X4 = df4 [[ 'Lines_Of_Program', 'Physic_Lines',
1525             'n1_Number_Of_Distinct_Operators',
1526             'n2_Number_Of_Distinct_Operands', 'n_Program_Vocabulary',
1527             'N1_Total_Number_Of_Operators', 'N2_Total_Number_Of_Operands',
1528             'N_Program_Length', 'B_Number_of_Delivered_Bugs_1',
1529             'B_Number_of_Delivered_Bugs_2',
1530             'D_Difficulty', 'E_Effort',
1531             'T_Time_Required_To_Program', 'V_Volume',
1532             '_N_Calculated_Program_Length', 'McCab_Number' ]].values
1533
1534
1535 y4= df4 [ 'IsVulnerable' ].values
1536
1537
1538
1539
1540
1541 X5 = df5 [[ 'Lines_Of_Program', 'Physic_Lines',
1542             'n1_Number_Of_Distinct_Operators',
1543             'n2_Number_Of_Distinct_Operands', 'n_Program_Vocabulary',
1544             'N1_Total_Number_Of_Operators',
1545             'N2_Total_Number_Of_Operands', 'N_Program_Length',
1546             'B_Number_of_Delivered_Bugs_1',
1547             'B_Number_of_Delivered_Bugs_2',
1548             'D_Difficulty', 'E_Effort',
1549             'T_Time_Required_To_Program', 'V_Volume',
1550             '_N_Calculated_Program_Length', 'McCab_Number' ]].values
1551
1552
1553 y5= df5 [ 'IsVulnerable' ].values
1554
1555
1556 print (len(y5))
1557
1558
1559 import numpy as np
1560 X=np.concatenate((X,X1,X2,X3,X4,X5))
1561 y=np.concatenate((y,y1,y2,y3,y4,y5))
1562
1563
1564
1565 # label_encoder object knows how to understand word labels.
1566 label_encoder = preprocessing.LabelEncoder()
1567
1568 # Encode labels in column 'species'.
1569 y = label_encoder.fit_transform(y)
1570
1571 counter = Counter(y)
1572 print(counter)
1573
1574 #ros = RandomOverSampler(sampling_strategy={ 0: 85399,1: 85399},
1575 random_state=42) # String
1576 #X, y = ros.fit_resample(X, y)
1577
1578 #rus = RandomUnderSampler(sampling_strategy={ 0: 3475,1:
1579 3475,2:3475,3:3475,4:3475,5:3475,6:3475}, random_state=42) # String
1580 #X, y = rus.fit_resample(X, y)
1581

```

```

1582 #smote = SMOTE() #SMOTE("minority")
1583 #X, y= smote.fit_resample(X, y)
1584
1585 rus = RandomUnderSampler(sampling_strategy={0:15000,1:
1586 9952,3:13603,2:10440,4:7285,5:3475,6:10926}, random_state=42) # String
1587 X, y = rus.fit_resample(X, y)
1588
1589
1590
1591 X_train, X_test, y_train, y_test = train_test_split(X,y,
1592 test_size=0.33,shuffle=True, random_state=42, stratify=y)
1593 X_test1=X_test
1594 X_train1=X_train
1595 y_test1=y_test
1596 y_train1=y_train
1597
1598
1599 from tensorflow.keras.utils import to_categorical
1600 y_train = to_categorical(y_train)
1601 y_test = to_categorical(y_test)
1602
1603
1604
1605 from sklearn.preprocessing import MinMaxScaler
1606 scaler = MinMaxScaler()
1607 scaler.fit(X_train)
1608
1609 Xtrain_scaled = scaler.transform(X_train)
1610 Xtest_scaled = scaler.transform(X_test)
1611
1612 Xtrain_scaled.shape
1613
1614
1615 num_classes = 7
1616
1617 def model_VGG16(learning_rate=0.001, momentum=0.9):
1618     model = Sequential()
1619
1620
1621     model.add(Conv1D(64, 2, activation="relu",
1622 input_shape=(X.shape[1],1)))
1623     model.add(Conv1D(64, 2, activation='relu',
1624 kernel_initializer='he_uniform',padding='same'))
1625     model.add(MaxPooling1D(pool_size=1))
1626
1627     model.add(Conv1D(128,2, activation='relu',
1628 kernel_initializer='he_uniform',padding='same'))
1629     model.add(Conv1D(128, 2, activation='relu',
1630 kernel_initializer='he_uniform',padding='same'))
1631     model.add(MaxPooling1D(pool_size=1))
1632
1633     model.add(Conv1D(256, 2, activation='relu',
1634 kernel_initializer='he_uniform',padding='same'))
1635     model.add(Conv1D(256, 2, activation='relu',
1636 kernel_initializer='he_uniform',padding='same'))
1637     model.add(MaxPooling1D(pool_size=1))
1638
1639     model.add(Conv1D(512,2, activation='relu',
1640 kernel_initializer='he_uniform',padding='same'))
1641     model.add(Conv1D(512, 2, activation='relu',
1642 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',

```

```

1823 kernel_initializer='he_uniform',padding='same'))
1824 model.add(Dropout(0.5))
1825
1826 model.add(Conv1D(128, 2, activation='relu',
1827 kernel_initializer='he_uniform',padding='same'))
1828 model.add(Conv1D(128,2, activation='relu',
1829 kernel_initializer='he_uniform', padding='same'))
1830 model.add(Conv1D(128, 2, activation='relu',
1831 kernel_initializer='he_uniform',padding='same'))
1832 model.add(Conv1D(128,2, activation='relu',
1833 kernel_initializer='he_uniform',padding='same'))
1834 model.add(Conv1D(128, 2, activation='relu',
1835 kernel_initializer='he_uniform',padding='same'))
1836 model.add(Conv1D(128,2, activation='relu',
1837 kernel_initializer='he_uniform',padding='same'))
1838 model.add(Conv1D(128, 2, activation='relu',
1839 kernel_initializer='he_uniform',padding='same'))
1840
1841 model.add(Conv1D(256, 2, activation='relu',
1842 kernel_initializer='he_uniform',padding='same'))
1843 model.add(Dropout(0.5))
1844
1845 model.add(Conv1D(256, 2, activation='relu',
1846 kernel_initializer='he_uniform',padding='same'))
1847 model.add(Conv1D(256, 2, activation='relu',
1848 kernel_initializer='he_uniform',padding='same'))
1849 model.add(Conv1D(256, 2, activation='relu',
1850 kernel_initializer='he_uniform',padding='same'))
1851 model.add(Conv1D(256, 2, activation='relu',
1852 kernel_initializer='he_uniform',padding='same'))
1853 model.add(Conv1D(256, 2, activation='relu',
1854 kernel_initializer='he_uniform',padding='same'))
1855 model.add(Conv1D(256, 2, activation='relu',
1856 kernel_initializer='he_uniform',padding='same'))
1857 model.add(Conv1D(256, 2, activation='relu',
1858 kernel_initializer='he_uniform',padding='same'))
1859 model.add(Conv1D(256, 2, activation='relu',
1860 kernel_initializer='he_uniform',padding='same'))
1861 model.add(Conv1D(256, 2, activation='relu',
1862 kernel_initializer='he_uniform', padding='same'))
1863 model.add(Conv1D(256, 2, activation='relu',
1864 kernel_initializer='he_uniform',padding='same'))
1865 model.add(Conv1D(256, 2, activation='relu',
1866 kernel_initializer='he_uniform',padding='same'))
1867
1868 model.add(Conv1D(512,2, activation='relu',
1869 kernel_initializer='he_uniform',padding='same'))
1870 model.add(Dropout(0.5))
1871
1872 model.add(Conv1D(512, 2, activation='relu',
1873 kernel_initializer='he_uniform',padding='same'))
1874 model.add(Conv1D(512, 2, activation='relu',
1875 kernel_initializer='he_uniform',padding='same'))
1876 model.add(Conv1D(512, 2, activation='relu',
1877 kernel_initializer='he_uniform',padding='same'))
1878 model.add(Conv1D(512,2, activation='relu',
1879 kernel_initializer='he_uniform',padding='same'))
1880 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 =

```



```

1943 compute_class_weight(
1944
1945             class_weight = "balanced",
1946             classes = np.unique(y_train_new),
1947             y = y_train_new
1948         )
1949 class_weights = dict(zip(np.unique(y_train_new), class_weights))
1950 class_weights
1951
1952
1953
1954
1955 history1 = Vuln_model1.fit(x= Xtrain_scaled ,
1956 y=y_train ,batch_size=128, epochs=50, verbose=1,
1957 validation_split=0.33,class_weight=class_weights)
1958 scores1 = Vuln_model1.evaluate(x= Xtest_scaled ,
1959 y=y_test , verbose=1)
1960 print('Final Vuln_model1(VGG16) accuracy:', scores1[1])
1961
1962
1963
1964
1965
1966 plt.plot(history1.history['accuracy'])
1967 plt.plot(history1.history['val_accuracy'])
1968
1969 plt.title('VGG16_model_accuracy')
1970 plt.ylabel('accuracy')
1971 plt.xlabel('epoch')
1972 plt.legend(['train', 'test'], loc='upper_left')
1973 plt.show()
1974
1975 plt.plot(history1.history['loss'])
1976 plt.plot(history1.history['val_loss'])
1977
1978 plt.title('VGG16_model_loss')
1979 plt.ylabel('loss')
1980 plt.xlabel('epoch')
1981 plt.legend(['train', 'test'], loc='upper_left')
1982 plt.show()
1983
1984
1985 classes = Vuln_model1.predict(x= Xtest_scaled)
1986 y_classes = classes.argmax(axis=-1)
1987 true_classes = y_test.argmax(axis=-1)
1988
1989 print(classification_report(true_classes, y_classes, digits=6))
1990
1991 cf_matrix = confusion_matrix(true_classes, y_classes)
1992 print(cf_matrix)
1993
1994 history2 = Vuln_model2.fit(x= Xtrain_scaled , y=y_train ,
1995 batch_size=128, epochs=20, verbose=1,
1996 validation_split=0.33,class_weight=class_weights)
1997 scores2 = Vuln_model2.evaluate(x= Xtest_scaled ,
1998 y=y_test , verbose=1)
1999 print('Final Vuln_model2(VGG19) accuracy:', scores1[1])
2000
2001 import matplotlib.pyplot as plt2
2002
2003

```

```

2004 plt2.plot(history2.history['accuracy'])
2005 plt2.plot(history2.history['val_accuracy'])
2006
2007 plt2.title('VGG19_model_accuracy')
2008 plt2.ylabel('accuracy')
2009 plt2.xlabel('epoch')
2010 plt2.legend(['train', 'test'], loc='upper_left')
2011 plt2.show()
2012
2013 plt2.plot(history2.history['loss'])
2014 plt2.plot(history2.history['val_loss'])
2015
2016 plt2.title('VGG19_model_loss')
2017 plt2.ylabel('loss')
2018 plt2.xlabel('epoch')
2019 plt2.legend(['train', 'test'], loc='upper_left')
2020 plt2.show()
2021
2022
2023 classes = Vuln_model2.predict(x= Xtest_scaled)
2024 y_classes = classes.argmax(axis=-1)
2025 true_classes = y_test.argmax(axis=-1)
2026
2027 print(classification_report(true_classes, y_classes, digits=6))
2028
2029
2030 cf_matrix = confusion_matrix(true_classes, y_classes)
2031 print(cf_matrix)
2032
2033 import matplotlib.pyplot as plt3
2034
2035
2036 history3 = Vuln_model3.fit(x= Xtrain_scaled, y=y_train,
2037 batch_size=128, epochs=20, verbose=1,
2038 validation_split=0.33, class_weight=class_weights)
2039 scores3 = Vuln_model3.evaluate(x= Xtest_scaled,
2040 y=y_test, verbose=1)
2041 print('Final_Vuln_model3_(AlexNet)_accuracy:', scores3[1])
2042
2043
2044
2045 plt3.plot(history3.history['accuracy'])
2046 plt3.plot(history3.history['val_accuracy'])
2047
2048 plt3.title('AlexNet_model_accuracy')
2049 plt3.ylabel('accuracy')
2050 plt3.xlabel('epoch')
2051 plt3.legend(['train', 'test'], loc='upper_left')
2052 plt3.show()
2053
2054 plt3.plot(history3.history['loss'])
2055 plt3.plot(history3.history['val_loss'])
2056
2057 plt3.title('AlexNet_model_loss')
2058 plt3.ylabel('loss')
2059 plt3.xlabel('epoch')
2060 plt3.legend(['train', 'test'], loc='upper_left')
2061 plt3.show()
2062
2063

```

```

2064 classes = Vuln_model3.predict(x= Xtest_scaled)
2065 y_classes = classes.argmax(axis=-1)
2066 true_classes = y_test.argmax(axis=-1)
2067 print(classification_report(true_classes , y_classes , digits=6))
2068
2069
2070
2071 cf_matrix = confusion_matrix(true_classes , y_classes)
2072 print(cf_matrix)
2073
2074
2075 import matplotlib.pyplot as plt4
2076
2077
2078 history4 = Vuln_model4.fit(x= Xtrain_scaled ,
2079 y=y_train , batch_size=128, epochs=20, verbose=1,
2080 validation_split=0.33,class_weight=class_weights)
2081 scores4 = Vuln_model4.evaluate(x= Xtest_scaled ,
2082 y=y_test , verbose=1)
2083 print( 'Final Vuln_model4(Resent) accuracy: ', scores4 [1])
2084
2085
2086 plt4.plot(history4.history['accuracy'])
2087 plt4.plot(history4.history['val_accuracy'])
2088
2089 plt4.title('Resent_model_accuracy')
2090 plt4.ylabel('accuracy')
2091 plt4.xlabel('epoch')
2092 plt4.legend(['train','test'], loc='upper_left')
2093 plt4.show()
2094
2095 plt4.plot(history4.history['loss'])
2096 plt4.plot(history4.history['val_loss'])
2097
2098 plt4.title('Resent_model_loss')
2099 plt4.ylabel('loss')
2100 plt4.xlabel('epoch')
2101 plt4.legend(['train','test'], loc='upper_left')
2102 plt4.show()
2103
2104
2105 classes = Vuln_model4.predict(x= Xtest_scaled)
2106 y_classes = classes.argmax(axis=-1)
2107 true_classes = y_test.argmax(axis=-1)
2108 print(classification_report(true_classes , y_classes , digits=6))
2109
2110
2111
2112 cf_matrix = confusion_matrix(true_classes , y_classes)
2113 print(cf_matrix)
2114
2115 history5 = Vuln_model5.fit(x= X_train , y=y_train ,
2116 batch_size=128, epochs=20, verbose=1,
2117 validation_split=0.33,class_weight=class_weights)
2118 scores5 = Vuln_model5.evaluate(x= X_test ,
2119 y=y_test , verbose=1)
2120 print( 'Final Vuln_model5(LSTM) accuracy: ', scores5 [1])
2121
2122 import matplotlib.pyplot as plt5
2123
2124 plt5.plot(history5.history['acc'])

```

```

2125 plt5.plot(history5.history['val_acc'])
2126
2127 plt5.title('LSTM_model_accuracy')
2128 plt5.ylabel('accuracy')
2129 plt5.xlabel('epoch')
2130 plt5.legend(['train', 'test'], loc='upper_left')
2131 plt5.show()
2132
2133 plt5.plot(history5.history['loss'])
2134 plt5.plot(history5.history['val_loss'])
2135
2136 plt5.title('LSTM_model_loss')
2137 plt5.ylabel('loss')
2138 plt5.xlabel('epoch')
2139 plt5.legend(['train', 'test'], loc='upper_left')
2140 plt5.show()
2141
2142
2143 classes = Vuln_model5.predict(x= X_test)
2144 y_classes = classes.argmax(axis=-1)
2145 true_classes = y_test.argmax(axis=-1)
2146
2147 print(classification_report(true_classes, y_classes, digits=6))
2148
2149
2150
2151 cf_matrix = confusion_matrix(true_classes, y_classes)
2152 print(cf_matrix)
2153
2154 group_names = ['True_Neg', 'False_Pos', 'False_Neg', 'True_Pos']
2155 group_counts = ["{0:0.0f}".format(value) for value in
2156                 cf_matrix.flatten()]
2157 group_percentages = ["{0:.2%}".format(value) for value in
2158                     cf_matrix.flatten()/np.sum(cf_matrix)]
2159 labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
2160           zip(group_names, group_counts, group_percentages)]
2161 labels = np.asarray(labels).reshape(2,2)
2162 sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
2163
2164
2165
2166
2167 Vuln_model1.save('/Users/abdullah/models/Model1.h5')
2168 Vuln_model2.save('/Users/abdullah/models/Model2.h5')
2169 Vuln_model3.save('/Users/abdullah/models/Model3.h5')
2170 Vuln_model4.save('/Users/abdullah/models/Model4.h5')
2171 Vuln_model5.save('/Users/abdullah/models/Model5.h5')
2172
2173
2174
2175 from keras.models import load_model
2176
2177 # load models from file
2178 def load_all_models(n_models):
2179     all_models = list()
2180     for i in range(n_models):
2181         # define filename for this ensemble
2182         filename = '/Users/abdullah/models/Model' + str(i+1) + '.h5'
2183         # load model from file
2184         model = load_model(filename)
2185         # add to list of members

```

```

2186         all_models.append(model)
2187         print('>loaded_\%s' \% filename)
2188     return all_models
2189
2190
2191 # load all models
2192 n_members = 5
2193 members = load_all_models(n_members)
2194 print('Loaded_\%d_models' \% len(members))
2195
2196
2197 # create stacked model input dataset as outputs from the ensemble
2198 def stacked_dataset(members, inputX):
2199     stackX = None
2200     for model in members:
2201         # make prediction
2202         yhat = model.predict(inputX, verbose=0)
2203         # stack predictions into [rows, members, probabilities]
2204         if stackX is None:
2205             stackX = yhat
2206         else:
2207             stackX = dstack((stackX, yhat))
2208     # flatten predictions to [rows, members x probabilities]
2209     stackX = stackX.reshape((stackX.shape[0],
2210 stackX.shape[1]*stackX.shape[2]))
2211     return stackX
2212
2213
2214
2215
2216 # fit a model based on the outputs from the ensemble members
2217 def fit_stacked_model(members, inputX, inputy):
2218     # create dataset using ensemble
2219     stackedX = stacked_dataset(members, inputX)
2220     # fit standalone model
2221     model = LogisticRegression()
2222     model.fit(stackedX, inputy)
2223     return model
2224
2225
2226 # fit stacked model using the ensemble
2227 from numpy import dstack
2228 from sklearn.linear_model import LogisticRegression
2229 y_test=np.argmax(y_test, axis=1)
2230
2231 model = fit_stacked_model(members, Xtest_scaled, y_test)
2232
2233
2234
2235 # make a prediction with the stacked model
2236
2237 def stacked_prediction(members, model, inputX):
2238     # create dataset using ensemble
2239     stackedX = stacked_dataset(members, inputX)
2240     # make a prediction
2241     yhat = model.predict(stackedX)
2242     return yhat
2243
2244 # evaluate model on test set
2245 from sklearn.metrics import accuracy_score
2246 yhat = stacked_prediction(members, model, Xtest_scaled)
2247
2248 acc = accuracy_score(y_test, yhat)

```

```

2249 print ( 'Stacked Test Accuracy: \%.3f' \% acc)
2250 print ( len ( yhat ))
2251
2252
2253 ', , '
2254
2255 y_classes = yhat.argmax(axis=-1)
2256 true_classes = y_test.argmax(axis=-1)
2257
2258 print ( yhat )
2259 print ( y_classes )
2260 print ( y_test )
2261 print ( true_classes )
2262
2263 for x in range ( len ( y_classes ) ):
2264     print ( classes [ x ] )
2265     print ( "++++++" )
2266     print ( true_classes )
2267 for x in range ( len ( true_classes ) ):
2268     ', , ' print ( true_classes [ x ] )
2269
2270 print ( classification_report ( y_test , yhat , digits = 6 ))
2271
2272 cf_matrix = confusion_matrix ( y_test , yhat )
2273 print ( cf_matrix )
2274
2275 group_names = [ 'True Neg' , 'False Pos' , 'False Neg' , 'True Pos' ]
2276 group_counts = [ "{0:0.0f}".format ( value ) for value in
2277                 cf_matrix.flatten () ]
2278 group_percentages = [ "{0:.2\%}" .format ( value ) for value in
2279                      cf_matrix.flatten () / np.sum ( cf_matrix ) ]
2280 labels = [ f "{v1}\n{v2}\n{v3}" for v1 , v2 , v3 in
2281            zip ( group_names , group_counts , group_percentages ) ]
2282 labels = np.asarray ( labels ).reshape ( 2 , 2 )
2283 sns.heatmap ( cf_matrix , annot = labels , fmt = ' ' , cmap = 'Blues' )

```

2284 REFERENCES

- 2285 Ferenc, R., Hegedűs, P., Gyimesi, P., Antal, G., Bán, D., and Gyimóthy, T. (2019). Challenging
2286 machine learning algorithms in predicting vulnerable javascript functions. In *2019 IEEE/ACM*
2287 *7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineer-*
2288 *ing (RAISE)*, pages 8–14. IEEE.
- 2289 Ganesh, S., Ohlsson, T., and Palma, F. (2021). Predicting security vulnerabilities using source
2290 code metrics. In *2021 Swedish Workshop on Data Science (SweDS)*, pages 1–7. IEEE.
- 2291 Ganesh, S., Palma, F., and Olsson, T. (2022). Are source code metrics “good enough” in
2292 predicting security vulnerabilities? *Data*, 7(9):127.
- 2293 Viszok, T., Hegedűs, P., and Ferenc, R. (2021). Improving vulnerability prediction of javascript
2294 functions using process metrics. *arXiv preprint arXiv:2105.07527*.