IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⋟ Impact Factor 8.066 ⋟ Peer-reviewed / Refereed journal ⋟ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

An Enhanced Stacking Ensemble Framework for Malware Detection in Android

Pragathi B[1], Aishwarya K[2], Pavana S[3], Shreya Shetty Suresh[4], Parvathi S J[5]

Student, Department of Computer Science and Engineering, GSSS Institute of Engineering and Technology for Women, Mysore, India Affiliated to Visvesvaraya Technological University, Belagavi Karnataka[1-4]

Associate Professor, Department of Computer Science and Engineering, GSSS Institute of Engineering and Technology for Women, Mysore, India Affiliated to Visvesvaraya Technological University, Belagavi Karnataka[5]

Abstract— Malware is nothing but the short name for malicious software, in general, referred to many forms of hostile or intrusion-creating software, spyware, Trojan horses, backdoor, and rootkits. The main aim of malware is to damage, steal, disrupt or do some bad actions. Malware is powerful enough to infect any kind of computing machine running applications, and the prevention of malware is well-studied for personal computers (PC). Smartphone device detection techniques used are lagging far behind compared to the fast growth of the mobile population being

Keywords— HTML, MySQL, PHP, Club Management, Web Server.

I. INTRODUCTION

The rapid growth of the Android ecosystem has brought about a significant increase in the number of mobile malware threats. Malicious applications targeting Android devices have become a persistent concern, posing serious risks to user privacy, data security, and overall system integrity. As a result, the development of effective malware detection techniques has become crucial to safeguarding the Android platform and its users.

Traditional malware detection approaches often rely on single classifiers, which may suffer from limitations such as high false positive rates and low detection accuracy. To address these challenges, we propose an enhanced stacking ensemble framework for malware detection in Android, aiming to leverage the strengths of multiple classifiers and enhance overall detection performance.

Need of web Application

The development of a web application for an enhanced stacking ensemble framework for malware detection in Android can bring several benefits, such as:

1.Improved accessibility: A web application can be accessed from anywhere with an internet connection, making it easy for users to access the malware detection framework without needing to install any software.
2.User-friendly interface: A well-designed web application can provide an intuitive user interface that makes it easy for users to interact with the malware detection framework.
3.Scalability: A web application can easily be scaled to handle a large number of users and can be deployed on multiple servers to handle heavy traffic.
4.Centralized management: A web application can provide centralized management of the malware detection framework, making it easier for administrators to manage and monitor the system.
5.Security: A web application can provide security features such as authentication, authorization, and encryption to protect the malware detection framework and the data it processes.
Overall, developing a web application for an enhanced stacking ensemble framework for malware detection in Android can improve the accessibility, usability, scalability, management, and security of the system, making it more effective and efficient in detecting and preventing malware on Android devices.

Need for club management

Club management can be beneficial in several ways for an enhanced stacking ensemble framework for malware detection in Android. Here are a few reasons why:

1.Collaboration: Club management can help bring together individuals with different backgrounds and expertise, allowing for more effective collaboration and knowledge sharing. This can be especially useful in developing a comprehensive and effective malware detection framework

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⁘ Impact Factor 8.066 ⁘ Peer-reviewed / Refereed journal ⁘ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

2. Resources: A club can provide access to resources that may not be available to an individual working alone. For example, the club may have access to specialized hardware or software tools that can aid in the development of the framework.

3. Feedback: Working in a club allows for more opportunities to receive feedback on the framework from different perspectives. This can help identify potential weaknesses or areas for improvement that may have been overlooked by an individual working alone.

4. Skill development: Club management can provide opportunities for members to develop new skills or refine existing ones. For example, members may have the opportunity to learn about different machine learning algorithms or programming languages, which can be applied to the development of the malware detection framework.

Overall, club management can be a valuable asset in the development of an enhanced stacking ensemble framework for malware detection in Android. By bringing together individuals with different backgrounds and expertise, providing access to resources, and facilitating collaboration and feedback, a club can help create a more effective and comprehensive framework

II. REQUIREMENTS

Software Requirements:

Server Side:

- OS: Windows XP or Higher

Development Side:

- OS: Windows XP or Higher

Front-End: Pycharm

Back-End: Google Colab

Client Side:

- OS: Windows XP or Higher

Hardware Requirements:

Server Side:

- Processor: Pentium 4+
- RAM: 2GB
- Hard Disk: 20 GB

User Side:

- Processor: Pentium 4+
- RAM: 2GB
- Hard Disk: 20 GB

III. IMPLEMENTATION

Flow chart

It is common practice to draw a context-level data flow diagram first, which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram (also known as the 'Level 0 DFD') the system's interactions with the outside world are modelled purely in terms of data flows across the system boundary. The context diagram shows the entire system as a single process and gives no clues as to its internal organization. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modelled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job and shows the flow of data between the various parts of the system

 This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⁜ Impact Factor 8.066 ⁜ Peer-reviewed / Refereed journal ⁜ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

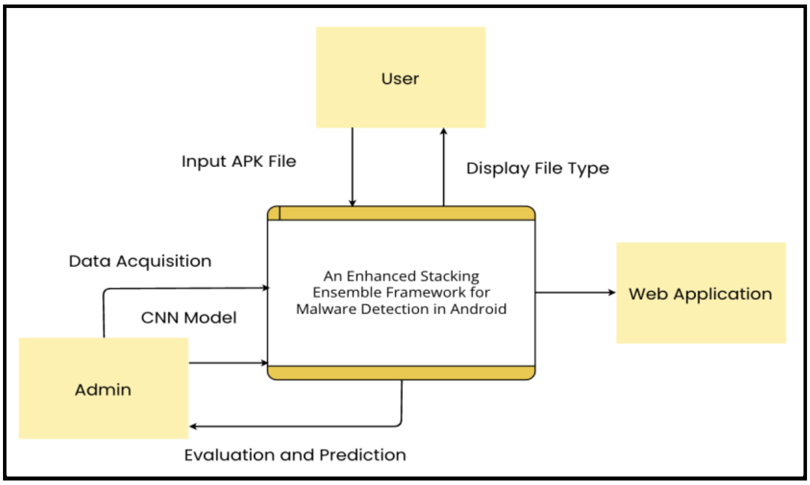

Figure-1: Flow chart of the system.

System architecture

The Android APK file can be decompressed to a series of resource files and code files, such as AndroidManifest.xml, Class.dex. Decompiling AndroidManifest.xml file can get the basic information of the application package, such as package name, version number, permissions, four major components and other information. Decompiling Class.dex files can generate a series of Smali files containing source code of the application package, such as the Android SDK API, third- party API and API call relations, etc. Because normal software and malware have different application preferences for Permission and API, malware often contains some Permissions and API that are not or very rare in normal software. Similarly, normal software also contains some Permission different from malware. Therefore, it is able to decide whether the application is malicious based on the appearance of Permission or API.

In addition to the Permission and API mentioned above, the static features that can be extracted include: the size, source, description information, category, and other attribute characteristics of the APK package itself; the signatures, abstracts, and other information in the META-INF file; the assets file of picture, audio and other resource information; four major components (Activity, Service, Broadcast Receiver, Content Provider), Intent Filter and other information. If fine-grained feature extraction is performed, a 1M-sized application package can have as many as 20,000 extractable features. Although this fine-grained feature extraction contains rich classification information, it is time-consuming and resource-consuming to process. The feature set often contains many repeated and non-meaningful features. In order to avoid the "curse of dimensionality" caused by high-dimensional features, the feature set can be reduced using dimensionality reduction or feature selection.

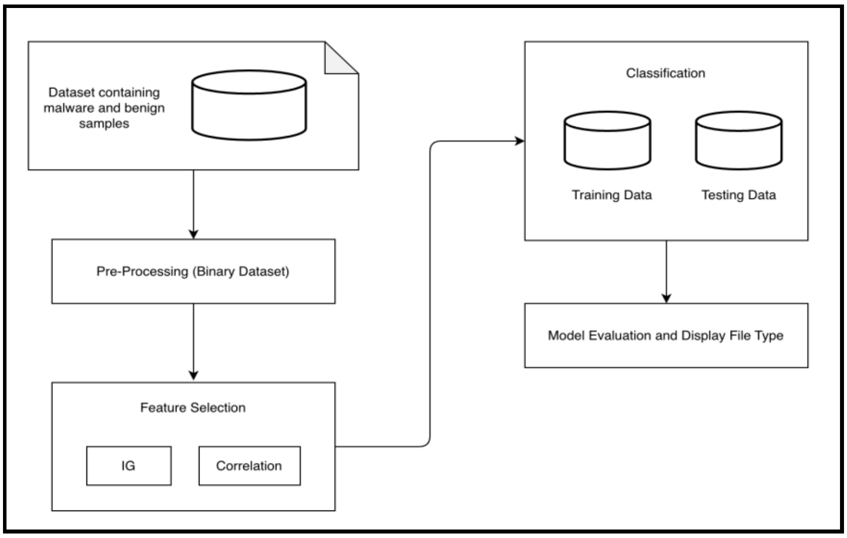

Figure-2: System architecture of the system.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⁕ Impact Factor 8.066 ⁕ Peer-reviewed / Refereed journal ⁕ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

- **Source Code**

Main connectivity:

```
from flask import Flask, render_template, request, redirect, url_for, flash
from werkzeug.utils import secure_filename
import os
import classifier
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = './static/upload/'
app.config['SECRET_KEY'] = 'd3Y5d5nJkU6CdwY'
if os.path.exists(app.config['UPLOAD_FOLDER']):
print("directory exists")
else:
os.makedirs(app.config['UPLOAD_FOLDER'])
print("directory created")
@app.route("/", methods=["GET", "POST"])
def home():
algorithms = {'Neural Network': '92.26 %', 'Support Vector Classifier': '89 %'}
result, accuracy, name, sdk, size = '', '', '', '', ''
if request.method == "POST":
if 'file' not in request.files:
flash('No file part')
return redirect(request.url)
file = request.files['file']
if file.filename == '':
flash('No selected file')
return redirect(request.url)
if file and file.filename.endswith('.apk'):
filename = secure_filename(file.filename)
print(filename)
file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
if request.form['algorithm'] == 'Neural Network':
accuracy = algorithms['Neural Network']
result, name, sdk, size = classifier.classify(os.path.join(app.config['UPLOAD_FOLDER'], filename), 0)
elif request.form['algorithm'] == 'Support Vector Classifier':
accuracy = algorithms['Support Vector Classifier']
result, name, sdk, size = classifier.classify(os.path.join(app.config['UPLOAD_FOLDER'], filename), 1)
return render_template("index.html", result=result, algorithms=algorithms.keys(), accuracy=accuracy, name=name,
                      sdk=sdk, size=size)
if __name__ == "__main__":
app.run(debug=True)
```

distribution connectivity:

```
import os
import glob
if os.name == 'nt':
try:
from ctypes import WinDLL
basedir = os.path.dirname(__file__)
except:
pass
else
libs_dir = os.path.abspath(os.path.join(basedir, '.libs'))
DLL_filenames = []
if os.path.isdir(libs_dir):
for filename in glob.glob(os.path.join(libs_dir,dll')):
WinDLL(os.path.abspath(filename))
DLL_filenames.append(filename)
if len(DLL_filenames) > 1:
import warnings
warnings.warn("loaded more than 1 DLL from .libs:""\n%s" % "\n".join(DLL_filenames),stacklevel=1)
```

Base connectivity:

```
  import copy
import warnings
```

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ჯ Impact Factor 8.066 ჯ Peer-reviewed / Refereed journal ჯ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

```
from collections import defaultdict
import platform
import inspect
import re
import numpy as np
from. import __version__
from ._config import get_config
from .utils import _IS_32BIT
from .utils._set_output import _SetOutputMixin
from .utils._tags import (
    _DEFAULT_TAGS,
)
from .utils.validation import check_X_y
from .utils.validation import check_array
from .utils.validation import _check_y
from .utils.validation import _num_features
from .utils.validation import _check_feature_names_in
from .utils.validation import _generate_get_feature_names_out
from .utils.validation import check_is_fitted
from .utils.validation import _get_feature_names
from .utils._estimator_html_repr import estimator_html_repr
from .utils._param_validation import validate_parameter_constraints
def clone(estimator, *, safe=True):
estimator_type = type(estimator)
if estimator_type in (list, tuple, set, frozenset):
return estimator_type([clone(e, safe=safe) for e in estimator])
elif not hasattr(estimator, "get_params") or isinstance(estimator, type):
if not safe:
return copy.deepcopy(estimator)
else:
if isinstance(estimator, type):
raise TypeError( "Cannot clone object. " + "You should provide an instance of " + "scikit-learn estimator instead of a class." )
else:
raise TypeError("Cannot clone object '%s' (type %s):  "it does not seem to be a scikit-learn ""estimator as it does not implement a
" "'get_params' method." % (repr(estimator), type(estimator)))
klass = estimator.__class__
new_object_params = estimator.get_params(deep=False)
for name, param in new_object_params.items():
new_object_params[name] = clone(param, safe=False)
new_object = klass(**new_object_params)
params_set = new_object.get_params(deep=False)
for name in new_object_params:
param1 = new_object_params[name]
param2 = params_set[name]
if param1 is not param2:
raise RuntimeError("Cannot clone object %s, as the constructor ""either does not set or modifies parameter %s" % (estimator,
name))
if hasattr(estimator, "_sklearn_output_config"):
new_object._sklearn_output_config = copy.deepcopy(estimator._sklearn_output_config)
return new_object
class BaseEstimator:
def _get_param_names(cls):
init = getattr(cls.__init__, "deprecated_original", cls.__init__)
if init is object.__init__:
return []
init_signature = inspect.signature(init)
parameters = [ p
    for p in init_signature.parameters.values()
    if p.name != "self" and p.kind != p.VAR_KEYWORD
]
for p in parameters:
if p.kind == p.VAR_POSITIONAL:
raise RuntimeError("scikit-learn estimators should always " "specify their parameters in the signature"" of their __init__ (no
varargs)." " %s with constructor %s doesn't " " follow this convention." % (cls, init_signature)  )
return sorted([p.name for p in parameters])
```

DOI: 10.17148/IARJSET.2023.105106

```
def get_params(self, deep=True):
out = dict()
for key in self._get_param_names():
value = getattr(self, key)
if deep and hasattr(value, "get_params") and not isinstance(value, type):
deep_items = value.get_params().items()
out.update((key + "__" + k, val) for k, val in deep_items)
out[key] = value
return out
def set_params(self,):
if not params:
return self
valid_params = self.get_params(deep=True)
nested_params = defaultdict(dict)  # grouped by prefix
for key, value in params.items():
key, delim, sub_key = key.partition("__")
if key not in valid_params:
local_valid_params = self._get_param_names()
raise ValueError(f"Invalid parameter {key!r} for estimator {self}. "f"Valid parameters are: {local_valid_params!r}.")
if delim:
nested_params[key][sub_key] = value
else:
 setattr(self, key, value)
valid_params[key] = value
for key, sub_params in nested_params.items():
if (key == "base_estimator"and valid_params[key] == "deprecated" and self.__module__.startswith("sklearn.")):
warnings.warn(f"Parameter 'base_estimator' of {self.__class__.__name__} is"" deprecated in favor of 'estimator'. See" f"
{self.__class__.__name__}'s docstring for more details." ,FutureWarning ,stacklevel=2,)
key = "estimator"
valid_params[key].set_params(**sub_params)
return self
def __repr__(self, N_CHAR_MAX=700):
from .utils._pprint import _EstimatorPrettyPrinter
N_MAX_ELEMENTS_TO_SHOW = 30
pp = _EstimatorPrettyPrinter(compact=True,indent=1, indent_at_name=True,
n_max_elements_to_show=N_MAX_ELEMENTS_TO_SHOW,)
repr_ = pp.pformat(self)
n_nonblank = len("".join(repr_.split()))
if n_nonblank > N_CHAR_MAX:
lim = N_CHAR_MAX
regex = r"^(\s*\S){%d}" % lim
left_lim = re.match(regex, repr_).end()
right_lim = re.match(regex, repr_[::-1]).end()
if "\n" in repr_[left_lim:-right_lim]:
regex += r"[^\n]*\n"
right_lim = re.match(regex, repr_[::-1]).end()
ellipsis = "..."
if left_lim + len(ellipsis) < len(repr_) - right_lim:
 repr_ = repr_[:left_lim] + "..." + repr_[-right_lim:]
return repr_
def __getstate__(self):
if getattr(self, "__slots__", None):
raise TypeError("You cannot use `__slots__` in objects inheriting from ""`sklearn.base.BaseEstimator`.")
try
 state = super().__getstate__()
if state is None:
state = self.__dict__.copy()
except AttributeError:
state = self.__dict__.copy()
if type(self).__module__.startswith("sklearn."):
return dict(state.items(), _sklearn_version=__version__)
else:
return state
def __setstate__(self, state):
if type(self).__module__.startswith("sklearn."):
```

```
pickle_version = state.pop("_sklearn_version", "pre-0.18")
if pickle_version != __version__:
warnings.warn(
format( self.__class__.__name__, pickle_version, __version__),UserWarning,)
try:
super().__setstate__(state)
except AttributeError:
self.__dict__.update(state)
def _more_tags(self):
return _DEFAULT_TAGS
def _get_tags(self):
collected_tags = {}
for base_class in reversed(inspect.getmro(self.__class__)):
if hasattr(base_class, "_more_tags"):
more_tags = base_class._more_tags(self)
collected_tags.update(more_tags)
return collected_tags
def _check_n_features(self, X, reset):
try:
n_features = _num_features(X)
except TypeError as e:
if not reset and hasattr(self, "n_features_in_"):
raise ValueError("X does not contain any features, but " f"{self.__class__.__name__} is expecting "f"{self.n_features_in_}
features") from e
return
if not hasattr(self, "n_features_in_"):
return
if n_features != self.n_features_in_:
raise ValueError(f"X has {n_features} features, but {self.__class__.__name__} "
        f"is expecting {self.n_features_in_} features as input."
    )
def _check_feature_names(self, X, *, reset):
if reset:
feature_names_in = _get_feature_names(X)
if feature_names_in is not None:
self.feature_names_in_ = feature_names_in
elif hasattr(self, "feature_names_in_"):
delattr(self, "feature_names_in_")
return
fitted_feature_names = getattr(self, "feature_names_in_", None)
X_feature_names = _get_feature_names(X)
if fitted_feature_names is None and X_feature_names is None:
return
if X_feature_names is not None and fitted_feature_names is None:
warnings.warn(f"X has feature names, but {self.__class__.__name__} was fitted without"" feature names")
return
if X_feature_names is None and fitted_feature_names is not None:
warnings.warn("X does not have valid feature names, but"f" {self.__class__.__name__} was fitted with feature names")
return
if len(fitted_feature_names) != len(X_feature_names) or np.any( fitted_feature_names != X_feature_names):
message = ("The feature names should match those that were passed during fit.\n")
fitted_feature_names_set = set(fitted_feature_names)
X_feature_names_set = set(X_feature_names)
unexpected_names = sorted(X_feature_names_set - fitted_feature_names_set)
missing_names = sorted(fitted_feature_names_set - X_feature_names_set)
def add_names(names):
output = ""
max_n_names = 5
for i, name in enumerate(names):
if i >= max_n_names:
output += "- ...\n"
break
output += f"- {name}\n"
return output
if unexpected_names:
```

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⁜ Impact Factor 8.066 ⁜ Peer-reviewed / Refereed journal ⁜ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

```
message += "Feature names unseen at fit time:\n"
message += add_names(unexpected_names)
if missing_names:
message += "Feature names seen at fit time, yet now missing:\n"
message += add_names(missing_names)
if not missing_names and not unexpected_names:
message += ("Feature names must be in the same order as they were in fit.\n" )
raise ValueError(message)
def _validate_data(self,X="no_validation", y="no_validation",reset=True, validate_separately=False,):
self._check_feature_names(X, reset=reset)
if y is None and self._get_tags()["requires_y"]:
raise ValueError(f"This {self.__class__.__name__} estimator ""requires y to be passed, but the target y is None.")
no_val_X = isinstance(X, str) and X == "no_validation"
no_val_y = y is None or isinstance(y, str) and y == "no_validation"
default_check_params = {"estimator": self}
check_params = {**default_check_params,}
if no_val_X and no_val_y:
raise ValueError("Validation should be done on X, y or both.")
elif not no_val_X and no_val_y:
X = check_array(X, input_name="X")
out = X
elif no_val_X and not no_val_y:
y = _check_y(y, **check_params)
out = y
else:
if validate_separately:
check_X_params, check_y_params = validate_separately
if "estimator" not in check_X_params:
check_X_params = {**default_check_params, }
X = check_array(X, input_name="X",)
if "estimator" not in check_y_params:
check_y_params = {**default_check_params, }
y = check_array(y, input_name="y",)
else:
X, y = check_X_y(X, y,)
out = X, y
if not no_val_X and check_params.get("ensure_2d", True):
self._check_n_features(X, reset=reset)
return out
def _validate_params(self):
validate_parameter_constraints(
self._parameter_constraints,
 self.get_params(deep=False),
caller_name=self.__class__.__name__,
)
def _repr_html_(self):
if get_config()["display"] != "diagram":
raise AttributeError("_repr_html_ is only defined when the ""'display' configuration option is set to ""'diagram'")
  return self._repr_html_inner
def _repr_html_inner(self):
return estimator_html_repr(self)
def _repr_mimebundle_(self,):
output = {"text/plain": repr(self)}
if get_config()["display"] == "diagram":
output["text/html"] = estimator_html_repr(self)
return output
class ClassifierMixin:
_estimator_type = "classifier"
def score(self, X, y, sample_weight=None):
from .metrics import accuracy_score
return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
def _more_tags(self):
return {"requires_y": True}
class RegressorMixin:
_estimator_type = "regressor"
```

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified ⋇ Impact Factor 8.066 ⋇ Peer-reviewed / Refereed journal ⋇ Vol. 10, Issue 5, May 2023
DOI: 10.17148/IARJSET.2023.105106

```
def score(self, X, y, sample_weight=None):
from .metrics import r2_score
y_pred = self.predict(X)
return r2_score(y, y_pred, sample_weight=sample_weight)
def _more_tags(self):
return {"requires_y": True}
class ClusterMixin:
_estimator_type = "clusterer”
def fit_predict(self, X, y=None):
self.fit(X)
return self.labels_
def _more_tags(self):
return {"preserves_dtype": []}
class BiclusterMixin:
def biclusters_(self):
return self.rows_, self.columns_
def get_indices(self, i):
rows = self.rows_[i]
columns = self.columns_[i]
return np.nonzero(rows)[0], np.nonzero(columns)[0]
def get_shape(self, i):
indices = self.get_indices(i)
return tuple(len(i) for i in indices)
def get_submatrix(self, i, data):
from .utils.validation import check_array
data = check_array(data, accept_sparse="csr")
row_ind, col_ind = self.get_indices(i)
return data[row_ind[:, np.newaxis], col_ind]
class TransformerMixin(_SetOutputMixin):
def fit_transform(self, X, y=None,):
if y is None:
return self.fit(X,).transform(X)
else:
return self.fit(X, y,).transform(X)
class OneToOneFeatureMixin:
def get_feature_names_out(self, input_features=None):
return _check_feature_names_in(self, input_features)
class ClassNamePrefixFeaturesOutMixin:
def get_feature_names_out(self, input_features=None):
check_is_fitted(self, "_n_features_out")
return _generate_get_feature_names_out(self, self._n_features_out, input_features=input_features)
class DensityMixin:
_estimator_type =
def score(self, X, y=None):
pass
class OutlierMixin:
_estimator_type = "outlier_detector"
def fit_predict(self, X, y=None):
return self.fit(X).predict(X)
class MetaEstimatorMixin:
_required_parameters = ["estimator"]
class MultiOutputMixin:
def _more_tags(self):
return {"multioutput": True}
class _UnstableArchMixin:
def _more_tags(self):
return {
"non_deterministic": ( _IS_32BIT or platform.machine().startswith(("ppc", "powerpc")))
}
def is_classifier(estimator):
return getattr(estimator, "_estimator_type", None) == "classifier"
def is_regressor(estimator):
return getattr(estimator, "_estimator_type", None) == "regressor"
def is_outlier_detector(estimator):
return getattr(estimator, "_estimator_type", None) == "outlier_detector"
```

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified ⁙ Impact Factor 8.066 ⁙ Peer-reviewed / Refereed journal ⁙ Vol. 10, Issue 5, May 2023
DOI: 10.17148/IARJSET.2023.105106

Classifier connectivity:

```
import os
import pickle
import numpy as np
from tensorflow import keras
from keras.models import load_model
from androguard.core.bytecodes.apk import APK
from genetic_algorithm import GeneticSelector
class CustomUnpickler(pickle.Unpickler):
def find_class(self, module, name):
try:
return super().find_class(__name__, name)
except AttributeError:
return super().find_class(module, name)
sel = CustomUnpickler(open('static/models/model-1.pkl', 'rb')).load()
permissions = []
with open('./static/permissions.txt', 'r') as f:
content = f.readlines()
for line in content:
cur_perm = line[:-1]
permissions.append(cur_perm)
def classify(file, ch):
vector = {}
result = 0
name, sdk, size = 'unknown', 'unknown', 'unknown'
app = APK(file)
perm = app.get_permissions()
name, sdk, size = meta_fetch(file)
for p in permissions:
if p in perm:
vector[p] = 1
else:
vector[p] = 0
 data = [v for v in vector.values()]
 data = np.array(data)
 if ch == 0:
ANN = load_model('static/models/model-3.h5')
result = ANN.predict([data[sel.support_].tolist()])
print(result)
if result < 0.02:
result = 'Benign (Safe)'
else:
result = 'Malware'
if ch == 1:
SVC = pickle.load(open('static/models/model-2.pkl', 'rb'))
result = SVC.predict([data[sel.support_]])
if result == 'benign':
result = 'Benign (Safe)'
else:
result = 'Malware
return result, name, sdk, size
def meta_fetch(apk):
app = APK(apk)
return app.get_app_name(), app.get_target_sdk_version(), str(round(os.stat(apk).st_size / (1024 * 1024), 2)) + ' MB'
```

Html connectivity:

```
<html>
<head>
<link href="{{ url_for('static', filename='css/bulma.min.css') }}" rel="stylesheet">
</head>
<body>
<nav class="navbar is-fixed-top is-dark">
<div class="navbar-brand">
<a class="navbar-item has-text-weight-bold is-size-4" href="#">
```

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified ⁒ Impact Factor 8.066 ⁒ Peer-reviewed / Refereed journal ⁒ Vol. 10, Issue 5, May 2023
DOI: 10.17148/IARJSET.2023.105106

```
</a>
</div>
</nav>
<div class="container" style="margin:25vh;padding:30px;position:fixed;">
<h3 class="is-size-5">APK Classification</h3>
<br>
<form enctype="multipart/form-data" method="POST">
<div class="field">
<label class="label">Algorithm</label>
 <div class="control">
<div class="select">
  <select class="selectpicker form-control" name="algorithm">
{% for algorithm in algorithms %}
 <option value="{{ algorithm }}">{{ algorithm }}</option>
 {% endfor %}
 </select>
 </div>
 </div>
 </div>
 <br>
 <div class="field">
 <label class="label"> Upload App</label>
<div class="control">
 <div class="file">
 <input name="file" type="file">
 </div>
 </div>
<br>
<input class="button is-primary is-outlined is-small" class="form-control" type="submit" value="Predict"/>
</div>
</form>
<div class="col ">
<div style="position:fixed;top:30vh;left:50vw;width:300px">
<h5 class="is-size-4">Output </h5>
<br>
 <h6 class="is-size-6">Predicted Class:   {{ result }} </h6>
<h6>Model Accuracy: {{ accuracy }}
 </h6>
 <hr>
 <h5 class="is-size-4">Metadata</h5>
 <br>
 <h6>App Name: {{ name }} </h6>
 <h6>Target SDK Version: {{ sdk }} </h6>
 <h6>File size: {{ size }} </h6>
 </div>
 </div>
</div>
</body>
</html>
```

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⋇ Impact Factor 8.066 ⋇ Peer-reviewed / Refereed journal ⋇ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

IV.SNAP SHOTS

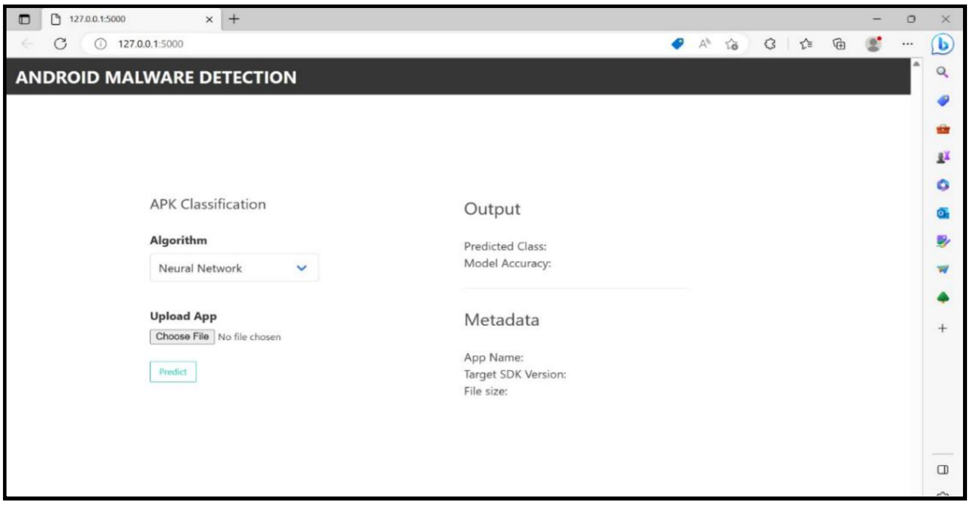

Figure-1: Home page

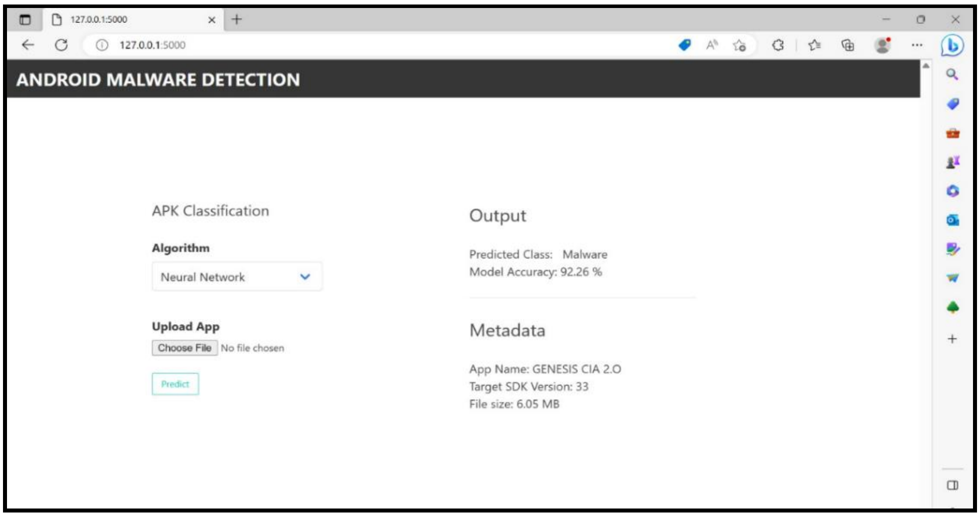

Figure-4: Detection of malware.

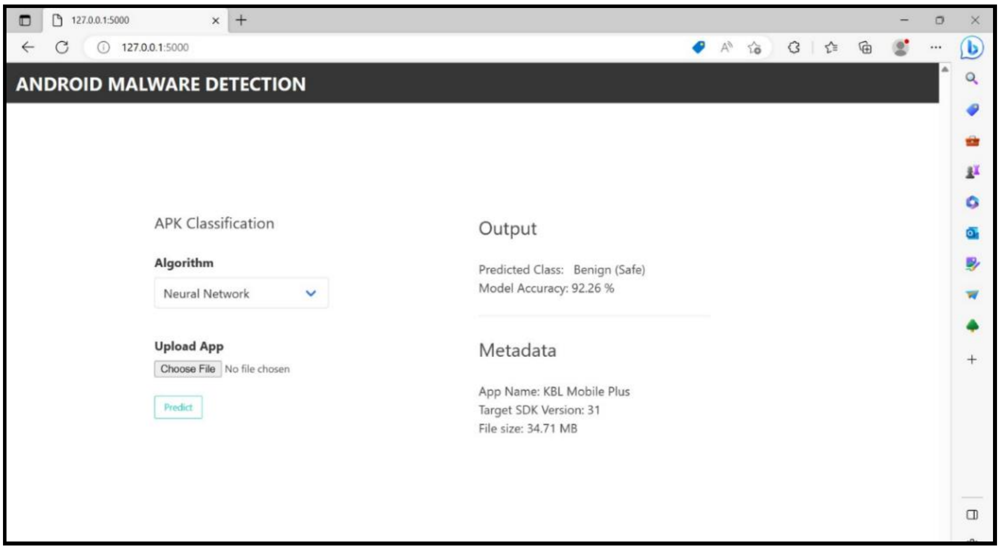

Figure-5: Detection of benign.

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

ISO 3297:2007 Certified ⁜ Impact Factor 8.066 ⁜ Peer-reviewed / Refereed journal ⁜ Vol. 10, Issue 5, May 2023

DOI: 10.17148/IARJSET.2023.105106

V.REFERENCES

[1] Anuar, Noor Azleen, Mohd Zaki Mas'ud, Nazrulazhar Bahaman, and Nor Azman Mat Ariff. "Analysis of machine learning classifier in Android malware detection through opcode." In 2020 IEEE Conference on Application, Information and Network Security (AINS), pp. 7-11. IEEE, 2020.

[2] Sandeep, H. R. "Static analysis of android malware detection using deep learning." In 2019 International Conference on Intelligent Computing and Control Systems (ICCS), pp. 841-845. IEEE, 2019.

[3] Karbab, ElMouatez Billah, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. "Android Malware Detection using Deep Learning" Digital Investigation 24 (2018): S48-S59.

[4] Chakravarty, Sujata. "Feature selection and evaluation of permission-based Android malware detection." In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184), pp. 795-799. IEEE, 2020.

[5] Rathore, Hemant, and Sanjay K. Sahay. "Towards Robust Android Malware Detection Models using Adversarial Learning." In 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pp. 424-425. IEEE, 2021.

VI.CONCLUSION

In conclusion, the development of an Enhanced Stacking Ensemble Framework for Malware Detection in Android presents a promising approach to combat the growing threat of malware in the Android ecosystem. By leveraging the power of ensemble learning and combining multiple base learners, this framework enhances the accuracy and robustness of malware detection compared to individual classifiers or standalone approaches.

Through the integration of diverse features, such as permissions, API calls, opcode sequences, and behavioural characteristics, the framework captures a comprehensive representation of Android applications, enabling effective discrimination between benign and malicious samples. The feature selection mechanisms further improve the efficiency and effectiveness of the ensemble framework by identifying the most informative features.

The evaluation results demonstrate the superior performance of the Enhanced Stacking Ensemble Framework in terms of detection accuracy, precision, recall, and other relevant metrics. Its ability to handle large-scale datasets, provide real-time or near-real-time analysis, and adapt to evolving malware threats showcases its scalability and flexibility.

The user-friendly interface and comprehensive reporting capabilities empower users to interact with the framework, submit Android applications for analysis, and gain insights into the detection results. Furthermore, the security and privacy considerations implemented in the framework ensure the protection of user data and maintain ethical practices.

However, it is important to note that no malware detection system is foolproof, and the evolving nature of malware poses ongoing challenges. Regular updates, model retraining, and adaptation to new malware samples are crucial to ensure the framework's effectiveness over time.

In conclusion, the Enhanced Stacking Ensemble Framework for Malware Detection in Android demonstrates significant potential in improving the accuracy, scalability, and efficiency of malware detection. Its successful implementation can contribute to strengthening the security of Android devices, protecting users from the growing threat landscape of Android malware.

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License