1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский государственный политехнический университет
3. —
4. Институт прикладной математики и механики
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 5**

1. **«Применение машинного обучения для выявления вредоносных файлов»**
2. по дисциплине «Безопасность операционных систем»
3. Выполнил
4. студент гр. 43609/3 Кожушок Д.И.

<*подпись*>

1. Проверил

преподаватель Жуковский Е.В.

<*подпись*>

1. Санкт-Петербург
2. 2018

# Цель работы

Изучение возможности применения машинного обучения в задачах, связанных с выявлением вредоносных файлов различного формата. Получение навыков построения классификаторов на основе распространенных методов машинного обучения.

# Задачи

1. Изучить формат файла, исследуемый в рамках лабораторной работы. Определить информацию, которая может выступать в качестве признаков для выявления вредоносных файлов.
2. Написать программу, осуществляющую парсинг формата файла и извлечение признаков из исследуемых файлов. В качестве признаков могут выступать значения полей файла, n-граммы, строки / ключевые слова (тэги, команды). Возможно использование существующих библиотек для работы с исследуемым типом файлов [4].
3. Осуществить формирование выборки вредоносных файлов с использованием различных подходов:
4. с использованием специализированных средств осуществить поиск в сети Интернет вредоносных сайтов, распространяющих файлы исследуемого типа [1];
5. поиск средств эксплуатации уязвимостей (вредоносные файлы исследуемого типа) в существующих базах уязвимостей и эксплойтов (например, exploid-db, компоненты Metasploit Framework);
6. поиск вредоносных файлов, исследуемого типа на специализированных ресурсах в сети Интернет, предназначенных для исследователей и аналитиков антивирусных лабораторий [2, 4].
7. Сформировать выборку из легитимных файлов, исследуемого типа.
8. Выборка должна быть представлена в виде таблицы признаков в формате CSV.
9. Провести анализ файлов, исследуемого типа и осуществить извлечение из них признаков с помощью разработанной программы.
10. Провести статистический анализ признаков для вредоносных и легитимных файлов. Определить наиболее характерные для вредоносных файлов признаки и построить диаграмму распределения значений данного признака. Сократить множество рассматриваемых признаков с использованием указанного в варианте задания подхода (критерий 𝜒2 / метод взаимной информации).
11. Построить классификатор вредоносных файлов. Рекомендуется использование библиотек Scikit-learn, Keras (Tensor Flow).
12. Провести тестирование разработанных классификаторов. Использовать различные виды перекрестной проверки (cross-validation): по K-блоков для K=5 / 10 / 20, случайное сэмплирование, поэлементная проверка [4].
13. Оценить эффективность работы построенных классификаторов. Составить таблицу с результатами тестирования (таблица 2) [5, 6]. Оценить точность (Precision), полноту (Recall) и F1-меру, построить AUC-ROC кривые. Посчитать процент ошибок 1-го и 2-го рода.

# Результаты

PDF (Portable Document Format) файл содержит четыре раздела:

* заголовок (header)
* тело файла (bode)
* таблица перекрестных ссылок (cross-reference table)
* поиск таблиц, объектов, ссылок (trailer)

Заголовок содержит номер версии.

Тело файла содержит последовательность косвенных объектов, входящих в состав публикации. Объекты – это компоненты публикации, такие как: страницы, изображения, шрифты.

Таблица перекрестных ссылок содержит информацию о каждом объекте в файле в виде одной строки описания места объекта в файле.

Trailer позволяет программному приложению, при чтении PDF файла, быстро находить таблицу перекрестных ссылок и специальные объекты.

Таким образом, была определена информация, которая может выступать в качестве признаков для выявления вредоносных файлов и разработана программа, которая собирает статистику этих признаков в файле.

Список собираемых признаков:

endobj

obj

endstream

stream

startxref

xref

trailer

/Pages

/PageLayout

/PageLabels

/PageMode

/Page

/FFilter

/Filter

/FDecodeParams

/First

/F

/Predictor

/Prev

/Pattern

/Perms

/PieceInfo

/Parent

/P

/ObjStm

/OpenAction

/Outlines

/O

/RichMedia

/Rows

/Root

/Resources

/Rotate

/R

/ColorTransform

/Colors

/Color

/EncryptMetadata

/Encrypt

/Kids

/K

/URI

/URLS

/U

/Names

/N

/Version

/VP

/V

/JS

/JavaScript

/AA

/AcroForm

/JBIG2Decode

/Launch

/EmbeddedFile

/XFA

/Length

/DL

/DecodeParams

/Columns

/EarlyChange

/BitsPerComponent

/EndOfLine

/EncodedByAlign

/Blackls1

/EndOfBlock

/DamagedRowsBeforeErrors

/JBIG2Globals

/Size

/ID

/Info

/Type

/Extends

/XRefStm

/Subfilter

/CF

/StmF

/StrF

/EFF

/Dests

/Extensions

/Lang

/Metadata

/SpiderInfo

/MarkInfo

/Legal

/Collection

/NeedRendering

/Count

/LastModified

/MediaBox

/CropBox

/BleedBox

/ArtBox

/TrimBox

/PZ

/Templates

/IDS

/XObject

Далее были сформированы две выборки файлов (легитимных и вредоносных). Размер легитимной выборки составляет 60 файлов, а вредоносных 100. Данные о файлах, после анализа программой, сохраняются в csv файле.

Далее был построен классификатор вредоносных файлов. Для алгоритма метод k-ближайших соседей осуществлялся выбор признаков методом χ2, а для нейронной сети методом взаимной информации.

Результат тестирования разработанных классификаторов представлены в таблице.

Таблица 1 – Результат тестирования разработанных классификаторов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вид перекрестной проверки | Применяемый алгоритм | TP, % | TN, % | FP, % | FN, % | Точность (Precision) | Полнота (Recall) | F1- мера | Время обучения | Время тестирования |
| По K-блоков, K = 5 | Метод k-ближайших соседей | 62 | 33 | 4 | 1 | 0.9815 | 0.3487 | 0.7746 | 0.00499 | 0.00699 |
| Нейронная сеть | 58 | 22 | 16 | 4 | 0.8333 | 0.2734375 | 0.6555377 | 2.196908 | 0.001999 |
| По K-блоков, K = 10 | Метод k-ближайших соседей | 62 | 33 | 4 | 1 | 0.9815 | 0.3487 | 0.6968 | 0.00402498 | 0.01199579 |
| Нейронная сеть | 58 | 26 | 12 | 4 | 0.8541666 | 0.30597 | 0.640215 | 4.77307 | 0.0009987 |
| По K-блоков, K = 20 | Метод k-ближайших соседей | 62 | 33 | 4 | 1 | 0.9818 | 0.3529 | 0.6467 | 0.014986 | 0.01599 |
| Нейронная сеть | 58 | 26 | 11 | 5 | 0.85714 | 0.311111 | 0.6128571 | 9.620908 | 0.00499868 |
| Поэлементная  выборка | Метод k-ближайших соседей | 62 | 35 | 2.5 | 0.5 | 0.9825 | 0.3613 | 0.61875 | 0.06797 | 0.083971 |
| Нейронная сеть | 58 | 27 | 11 | 4 | 0.86 | 0.316176 | 0.58125 | 82.8865356 | 0.030997038 |
| Случайное сэмплирование | Метод k-ближайших соседей | 75 | 22.5 | 2.5 | 0 | 1.0 | 0.230769 | 0.9836 | 0.0010185 | 0.00302076 |
| Нейронная сеть | 65 | 27.5 | 2.5 | 5 | 0.84615 | 0.297297 | 0.945454 | 0.429656 | 0.0058986 |

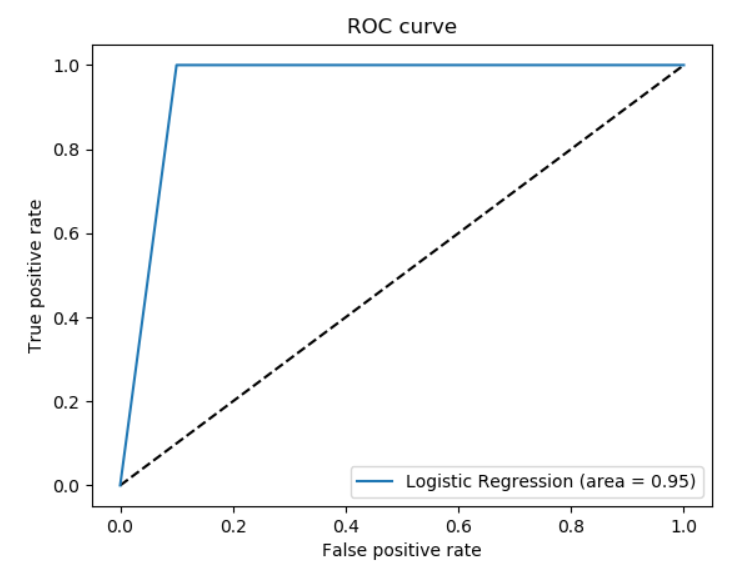


Рисунок 1 – ROC-кривая для метода k-ближайших соседей

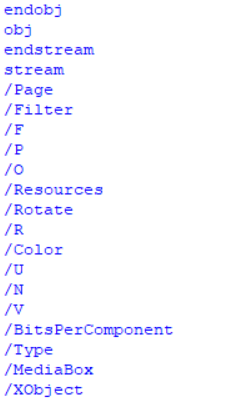


Рисунок 2 – Параметры выбранные с помощью метода χ2

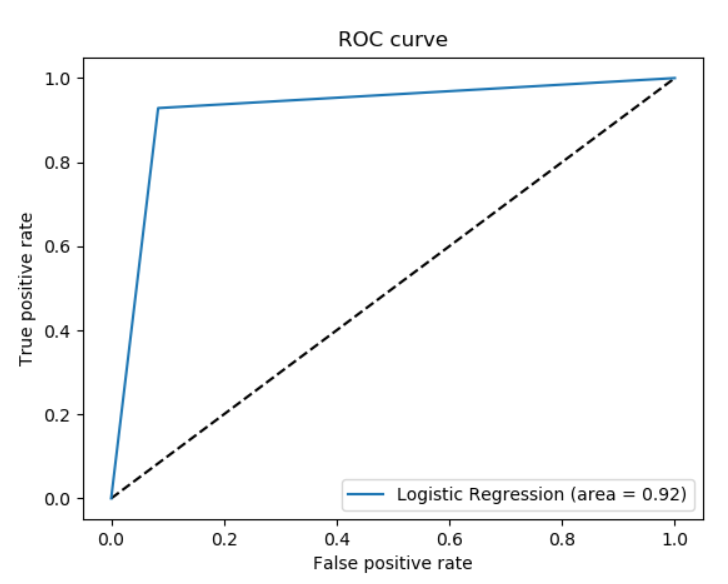


Рисунок 3 – ROC-кривая для алгоритма нейронная сеть

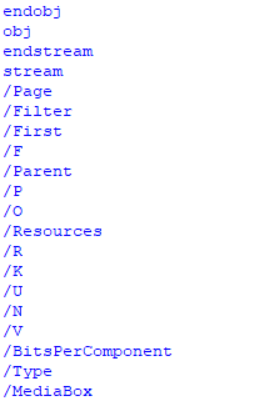


Рисунок 4 - Параметры выбранные с помощью метода взаимной информации

# Вывод

В результате лабораторной работы было получено две программы: парсер pdf файлов и классификатор. Было реализовано два алгоритма классификатора: алгоритм k-ближайших соседей и нейронная сеть. Также два вида критериев по которым будет определятся является ли файл вредоносным или легитимным: χ2 и метод взаимной информации.

При оценке эффективности классификаторов было выявлено, что наилучшие показатели достигаются при использовании алгоритма машинного обучения k-ближайших соседей. Точность достигается более 98%, площадь под ROC-кривой составляет 0,95, а также наилучшее время обучения. Так как выборки небольшие, то время обучения не сильно заметно, но при увеличении их это сильно скажется на времени обучения, что будет очень сильно заметно при использовании нейронной сети. Точность нейронной сети стартует от 83% и достигает наилучшего результата 86% при поэлементной проверке, а так же площадь под ROC-кривой составляет 0,92.

# Листинг

Парсер

import os

import re

import csv

keywords = ("endobj","obj","endstream","stream",

"startxref","xref", "trailer",

"/Pages","/PageLayout","/PageLabels","/PageMode","/Page",

"/FFilter","/Filter", "/FDecodeParams","/First","/F",

"/Predictor","/Prev","/Pattern","/Perms ","/PieceInfo","/Parent ","/P",

"/ObjStm","/OpenAction","/Outlines","/O",

"/RichMedia","/Rows","/Root","/Resources","/Rotate ","/R",

"/ColorTransform","/Colors","/Color",

"/EncryptMetadata","/Encrypt","/Kids","/K",

"/URI","/URLS","/U",

"/Names","/N","/Version","/VP","/V",

"/JS","/JavaScript","/AA","/AcroForm","/JBIG2Decode",

"/Launch","/EmbeddedFile","/XFA",

"/Lenght","/DL","/DecodeParams",

"/Columns","/EarlyChange","/BitsPerComponent",

"/EndOfLine","/EncodedByAlign",

"/Blackls1","/EndOfBlock","/DamagedRowsBeforeErrors","/JBIG2Globals",

"/Size","/ID","/Info","/Type", "/Extends","/XRefStm","/Subfilter",

"/CF","/StmF","/StrF","/EFF","/Dests","/Extensions",

"/Lang","/Metadata","/SpiderInfo ",

"/MarkInfo","/Legal","/Collection ","/NeedRendering",

"/Count ","/LastModified",

"/MediaBox","/CropBox ","/BleedBox ","/ArtBox",

"/TrimBox","/PZ ","/Templates","/IDS","/XObject")

#анализ файла

def statistics(file):

try:

infile = open(file, 'rb')

data=infile.read()

infile.close()

count=[]

#print(file)

#print(''.join([chr(byte) for byte in data]))

str\_dat=''.join([chr(byte) for byte in data])

for word in keywords:

words=len(re.findall(word, str\_dat))

if word == "obj":

words-=count[0]

elif word == "stream":

words-=count[2]

elif word =="/Page":

words = words - count[7]-count[8]-count[9]-count[10]

elif word == "xref":

words-=count[4]

elif word == "/F":

words=words-count[12]-count[13]-count[14]-count[15]

elif word == "/P":

words = words - count[7]-count[8]-count[9]-count[10]-count[11]-count[17]-count[18]-count[19]-count[20]-count[21]-count[22]

elif word =="/O":

words = words - count[24]-count[25]-count[26]

elif word=="/R":

words = words - count[28]-count[29]-count[30]-count[31]-count[32]

elif words =="/Color":

words = words - count[34]-count[35]

elif word =="/Encrypt":

words-=count[37]

elif words =="/K":

words-=count[39]

elif words =="/U":

words=words-count[41]-count[42]

elif words =="/N":

words-=count[44]

elif words =="/V":

words=words-count[46]-count[47]

count.append(words)

#print(word,words)

except:

return []

return count

#получаем список файлов

def analyze():

path="D:\\files\_5"

csva="D:\\parse.csv"

csvf="D:\\flag.csv"

file=open(csva , "w" , newline = "")

fileflag=open(csvf,"w",newline="")

#with open(csva , "w" , newline = "") as file:

# with open(csvf,"w",newline="")as fileflag:

wr=csv.writer(file,delimiter=';')

wf=csv.writer(fileflag,delimiter=';')

wr.writerow(keywords)

for root, dirs, files in os.walk(path):

for filename in files:

if filename.find('.pdf')!=-1:

st=statistics(root+'\\'+filename)

if st!=[]:

wr.writerow(st)

if root.find('malware')!=-1:

wf.writerow([1])

else:

wf.writerow([0])

print(root+'\\'+filename)

if \_\_name\_\_ == '\_\_main\_\_':

analyze()

Классификатор

import numpy as np

import os

import csv

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import mutual\_info\_classif

from sklearn.feature\_selection import chi2

from sklearn.neural\_network import MLPClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import f1\_score

from sklearn.model\_selection import KFold

from sklearn.preprocessing import minmax\_scale

from sklearn.model\_selection import LeaveOneOut

from sklearn.model\_selection import train\_test\_split

import time

import warnings

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import roc\_auc\_score, auc

from sklearn.metrics import roc\_curve

def draw\_auc(y, y\_pred):

fpr, tpr, thresholds = roc\_curve(y, y\_pred)

roc\_auc = auc(fpr, tpr)

plt.figure(1)

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % roc\_auc)

plt.xlabel('False positive rate')

plt.ylabel('True positive rate')

plt.title('ROC curve')

plt.legend(loc='best')

plt.show()

def cross\_validation(lenOfBlock , X , Y , num):

warnings.filterwarnings('ignore')

if num ==2:

clf = MLPClassifier(max\_iter=500, alpha=1.0, random\_state=21,tol=0.000000001)

elif num == 1:

clf = KNeighborsClassifier(n\_neighbors=3)

cv = KFold(n\_splits = lenOfBlock, random\_state = None, shuffle = False)

TN = 0

FP = 0

FN = 0

TP = 0

F1 = 0

Educate = 0.0

Test = 0.0

for train\_index, test\_index in cv.split(X):

start\_time = time.time()

clf.fit(X[train\_index], Y[train\_index])

education\_time = time.time() - start\_time

start\_time = time.time()

proba = clf.predict(X[test\_index])

test\_time = time.time() - start\_time

Educate += education\_time

Test += test\_time

tn, fp, fn, tp = confusion\_matrix(Y[test\_index], proba ,labels=[0,1]).ravel()

TN += tn

FP += fp

FN += fn

TP += tp

F1 += (f1\_score(Y[test\_index], proba , average='binary') )

summ = TN + FP + FN + TP

print('TP: ', TP / summ)

print('TN: ', TN / summ)

print('FP: ', FP / summ)

print('FN: ', FN / summ)

print('Точность (Precision): ', TN / (TN + FN))

print('Полнота(Recall)', TN / (TN + TP))

print('F-мера: ', F1 / lenOfBlock)

print('Время обучения: ', Educate)

print('Время тестирования: ', Test)

def element\_check(X , Y, num):

if num ==2:

clf = MLPClassifier(max\_iter=500, alpha=1.0, random\_state=21,tol=0.000000001)

elif num == 1:

clf = KNeighborsClassifier(n\_neighbors=3)

TN = 0

FP = 0

FN = 0

TP = 0

F1 = 0

Educate = 0.0

Test = 0.0

count = 0

loo = LeaveOneOut()

loo.get\_n\_splits(X)

for train\_index, test\_index in loo.split(X):

start\_time = time.time()

clf.fit(X[train\_index], Y[train\_index])

education\_time = time.time() - start\_time

start\_time = time.time()

proba = clf.predict(X[test\_index])

test\_time = time.time() - start\_time

Educate += education\_time

Test += test\_time

tn, fp, fn, tp = confusion\_matrix(Y[test\_index], proba ,labels=[0,1]).ravel()

TN += tn

FP += fp

FN += fn

TP += tp

count += 1

F1 += (f1\_score(Y[test\_index], proba , average='binary') )

summ = TP + TN + FP + FN

print('TP: ', TP / summ)

print('TN: ', TN / summ)

print('FP: ', FP / summ)

print('FN: ', FN / summ)

print('Точность (Precision): ', TN / (TN + FN))

print('Полнота(Recall)', TN / (TN + TP))

print('F-мера: ', F1 / len(Y))

print('Время обучения: ', Educate)

print('Время тестирования: ', Test)

def random\_sampling(X , Y, num):

tn = 0

fp = 0

fn = 0

fp = 0

education\_time = 0.0

test\_time = 0.0

if num == 2:

clf = MLPClassifier(max\_iter=500, alpha=1.0, random\_state=21,tol=0.000000001)

elif num ==1:

clf = KNeighborsClassifier(n\_neighbors=3)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y)

start\_time = time.time()

clf.fit(X\_train, y\_train)

education\_time = time.time() - start\_time

start\_time = time.time()

proba = clf.predict(X\_test)

test\_time = time.time() - start\_time

draw\_auc(y\_test, proba)

tn, fp, fn, tp = confusion\_matrix(y\_test, proba,labels=[0,1]).ravel()

summ = tn + fp + fn + tp

print('TP: ', tp / summ)

print('TN: ', tn / summ)

print('FP: ', fp / summ)

print('FN: ', fn / summ)

print('Точность (Precision): ', tn / (tn + fn))

print('Полнота(Recall)', tn / (tn + tp))

print('F-мера: ', f1\_score(y\_test, proba, average='binary'))

print('Время обучения: ', education\_time)

print('Время тестирования: ', test\_time)

#метод взаимной информации

def mutual\_info(data, labels):

selecter = SelectKBest(score\_func=mutual\_info\_classif, k=20)

selecter.fit(data, labels)

string = selecter.get\_support()

return selecter.transform(data),string

#критерий хи2

def method\_chi2(data, labels):

selecter = SelectKBest(score\_func=chi2, k=20)

selecter.fit(data, labels)

string = selecter.get\_support()

return selecter.transform(data),string

def read\_data():

data = []

names=[]

labels=[]

flag=0

with open('D:\\parse.csv') as file:

for line in file:

if flag ==0:

names=line.split(';')

flag=1

else:

parameters = line.split(';')

for i in range(len(parameters)):

parameters[i] = int(parameters[i])

data.append(parameters)

with open('D:\\flag.csv') as file:

for line in file:

labels.append(int(line[0]))

return np.array(data),names,labels

def main():

dataset,names,labels = read\_data()

print("1.Критерий x^2 и метод k-ближайших соседей\n2.Метод взаимной информации и нейронная сеть")

num=int(input())

if num == 1:

dataset,string = method\_chi2(dataset, labels)

for i in range(0,len(string)):

if string[i]==True:

print(names[i])

elif num == 2:

dataset,string = mutual\_info(dataset, labels)

for i in range(0,len(string)):

if string[i]==True:

print(names[i])

warnings.filterwarnings('ignore')

buffer\_test = minmax\_scale(dataset,feature\_range=(0, 1),axis = 0)

nptraining = np.array(buffer\_test , "float32")

nptarget = np.array(labels, "float32")

print("Перекрестная проверка k = 5:")

cross\_validation(5 , nptraining , nptarget,num)

print("Перекрестная проверка k = 10:")

cross\_validation(10 , nptraining , nptarget,num)

print("Перекрестная проверка k = 20:")

cross\_validation(20 , nptraining , nptarget,num)

print("Случайное сэмплирование:")

random\_sampling(nptraining , nptarget,num)

print("Поэлементная проверка:")

element\_check(nptraining , nptarget,

)

main()