ДОДАТОК А

Код програми

Utils.py

```
import functools
import collections

from operator import itemgetter
from itertools import filterfalse
from heapq import nsmallest


class IterCount(dict):

    def __missing__(self, k):
        return 0


def queues_stash(maxsize=100):
    maxqueuesize = maxsize * 10

    def fun_decor(u_func, length=len, iterator=iter, u_tuple=tuple, is_sorted=sorted, error_key=KeyError):
        user_stash = {}
        user_q = collections.deque()
        ref_count = IterCount()
        guard = object()
        mark = object()

        append_q, popleft_q = user_q.append, user_q.popleft
        appendleft_q, pop_q = user_q.appendleft, user_q.pop

        @functools.wraps(u_func)
        def common_wrap(*arglist, **kwdlist):
            value = arglist
            if kwdlist:
                value += (mark,) + u_tuple(is_sorted(kwdlist.items()))
            append_q(value)
            ref_count[value] += 1
            try:
                res = user_stash[value]
                common_wrap.hits += 1
            except error_key:
                res = u_func(*arglist, **kwdlist)
                user_stash[value] = res
                common_wrap.misses += 1

                if length(user_stash) > maxsize:
                    value = popleft_q()
                    ref_count[value] -= 1
                    while ref_count[value]:
                        value = popleft_q()
                        ref_count[value] -= 1
                    del user_stash[value], ref_count[value]
            if length(user_q) > maxqueuesize:
                ref_count.clear()
                appendleft_q(guard)
                for value in filterfalse(ref_count.__contains__,
                                         iterator(pop_q, guard)):
                    appendleft_q(value)
                    ref_count[value] = 1

            return res
```

```
        def clear_stash():
            user_stash.clear()
            user_q.clear()
            ref_count.clear()
            common_wrap.hits = common_wrap.misses = 0

        common_wrap.hits = common_wrap.misses = 0
        common_wrap.clear = clear_stash
        return common_wrap

    return fun_decor


def f_stash(maxsize=100):

    def decor_fun(user_func):
        user_stash = {}
        iter_count = IterCount()
        mark = object()

        @functools.wraps(user_func)
        def common_wrap(*arglist, **kwdlist):
            k = arglist
            if kwdlist:
                k += (mark,) + tuple(sorted(kwdlist.items()))
            iter_count[k] += 1


            try:
                res = user_stash[k]
                common_wrap.hits += 1
            except KeyError:
                res = user_func(*arglist, **kwdlist)
                user_stash[k] = res
                common_wrap.misses += 1

                if len(user_stash) > maxsize:
                    for k, _ in nsmallest(maxsize // 10,
                                          iter_count.iteritems(),
                                          key=itemgetter(1)):
                        del user_stash[k], iter_count[k]

            return res

        def clear():
            user_stash.clear()
            iter_count.clear()
            common_wrap.hits = common_wrap.misses = 0

        common_wrap.hits = common_wrap.misses = 0
        common_wrap.clear = clear
        return common_wrap

    return decor_fun
```

```
if __name__ == '__main__':

    @queues_stash(maxsize=20)
    def func(x, y):
        return 4 * x + y


    domain_u = range(5)
    from random import choice

    for i in range(1500):
        r = func(choice(domain_u), choice(domain_u))

    print(func.hits, func.misses)


    @f_stash(maxsize=20)
    def fun(x, y):
        return 4 * x + y


    domain_u = range(5)
    from random import choice

    for i in range(1500):
        r = fun(choice(domain_u), choice(domain_u))

    print(fun.hits, fun.misses)
```

NeuralNetwork.py

```
import logging
import re
import time
from collections import namedtuple

import numpy as np
from six.moves.cPickle import dump, load
from itertools import chain
from itertools import permutations
from pybrain.datasets import ClassificationDataSet
from pybrain.structure.modules import SoftmaxLayer, SigmoidLayer, LinearLayer
from pybrain.supervised.trainers import BackpropTrainer, RPropMinusTrainer
from pybrain.tools.shortcuts import buildNetwork
from pybrain.tools.customxml.networkwriter import NetworkWriter
from pybrain.utilities import percentError
from urllib.parse import urlparse, parse_qs
from multiprocessing import Pool
import matplotlib.pyplot as plt

from backports import f_stash

LogEntry = namedtuple('LogEntry', 'ip url code size refer useragent')
log = logging.getLogger('')
log.setLevel(logging.DEBUG)


def normal_request_func(req):
    vect_list = []
    if req == '-':
        return ['__REQUEST_WITHOUT_DATA__']
```

```
    try:
        method_name, url_val, http = req.split()
        vect_list.append('__NAME_OF_METHOD_' + method_name)
        vect_list.extend(map(lambda x: "__URL_VALUE_" + x, normal_given_url(url_val)))
        vect_list.append('__HTTP_VERSION_' + http)
    except Exception as exception:
        log.debug("This request is broken: {0}. Exception: {1}".format(req, exception))
        return ['__BROKEN_REQUEST__']
    return map(lambda i: "__REQUEST_" + i, vect_list)


@f_stash(maxsize=20000)
def normal_given_url(url):
    vect_list = []
    url_parsed = urlparse(url)
    vect_list.append('__SCHEME_' + url_parsed.scheme)
    vect_list.append('__NETLOC_' + url_parsed.netloc)
    if len(url_parsed.path) > 130:
        log.error("Url value is too long: {0}".format(url_parsed.path))
        vect_list.append('__URL_PATH_IS_TOO_LONG')
    else:
        vect_list.append('__URL_PATH_' + url_parsed.path)
    if url_parsed.query:
        vect_list.extend(map(lambda i: "__KEYS_QS_" + i, norm_keys_qs(url_parsed.query)))
    return vect_list


def norm_keys_qs(given_qs):
    keys_qs_list = parse_qs(given_qs).keys()
    if len(keys_qs_list) < 8:
        return parse_qs(given_qs).keys()
    else:
        log.debug("Keys list in qs is too long: {0}".format(given_qs))
        return ['QS_TOO_MANY_KEYS']


@f_stash(maxsize=20000)
def normal_reference(reference):
    if reference == '-':
        return ['__NO_REFERENCE__']
    return map(lambda x: "__REFERENCE_" + x, normal_given_url(reference))


@f_stash(maxsize=20000)
def normal_user_agent(user_agent):
    user_agent_regex = re.compile(r'(?P<comp>[^ ]+)(?:\s+\((?P<os>[^)]+)\)(?:\s+(?P<version>.*))?)?')
    if user_agent == '-':
        return ['__USER_AGENT_IS_EMPTY']
    try:
        parsed_user_agent = user_agent_regex.match(user_agent).groups()
    except Exception as exception:
        log.debug("Broken User Agent: {0}".format(exception))
        return ['__USER_AGENT_BROKEN']
    try:
        base_name, os_name, vers_val = parsed_user_agent
        vector_set = set()
        vector_set.add('__BASE_' + base_name)
        if not all([os_name, vers_val]):
            return ['__USER_AGENT_SIMPLE', '__USER_AGENT_ONLY_BASE_' + base_name]
        if not vers_val:
            vector_set.add('__THERE_IS_NO_VERSION')
        vector_set |= set(map(lambda i: '__OS_' + i.strip(), os_name.split(';')))
        vector_set |= set(map(lambda i: '__VERSION' + i.strip(' ()'), vers_val.split()))
        return map(lambda i: "__USER_AGENT_" + i, vector_set)
```

```
    except Exception as exception:
        log.debug("Unable parse User Agent: {0}".format(exception))
        return ['__USER_AGENT_IS_BROKEN']


@f_stash(maxsize=200000)
def entry_parsing(entry):
    entry_request = set(normal_request_func(entry.url))
    entry_refer = set(normal_reference(entry.refer))
    entry_ua = set(normal_user_agent(entry.useragent))
    try:
        entry_code = set()
        if int(entry.code) in [404, 403, 503]:
            entry_code.add('__ENTRY_CODE_' + entry.code)
    except Exception as e:
        log.debug("Unable parse request. Code: {0}".format(e))
        pass
    return entry_request | entry_refer | entry_ua | entry_code


def get_entry_vector(dict, entry_req):
    return np.array([i in entry_parsing(entry_req) for i in dict])


def add_samples_to_training_set(training_set, file_name, label):
    with open(file_name) as file_:
        for line in file_:
            try:
                entry = LogEntry(*nginx_log_re.match(line).groups())
                training_set.addSample(list(get_entry_vector(dictionary, entry)), label)
            except Exception:
                log.error('Failed to parse line: {0}'.format(line), exc_info=True)


def main_prog(log_name):
    start = time.time()
    nginx_log_re = re.compile(
        r'(?P<ip>[0-9.:a-f]+) [^ ]+ [^ ]+ \[.*\] "(?P<url>.*)" (?P<code>[0-9]+) (?P<size>[0-9]+|-) "(?P<refer>.*)" "(?P<useragent>.*)"$')

    log.warning('Preparing dictionary')
    dictionary = set()
    with open("good_file") as good_file:
        with open("bad_file") as bad_file:
            for line in chain(good_file, bad_file):
                try:
                    entry = LogEntry(*nginx_log_re.match(line).groups())
                    dictionary |= entry_parsing(entry)
                except Exception:
                    log.error('Failed to parse line: {0}'.format(line), exc_info=True)
    log.warning('Feature vector size: {0}'.format(len(dictionary)))
    dump(dictionary, open('dictionary.p', 'wb'))

    log.warning('Adding Samples')
    alldata = ClassificationDataSet(len(dictionary), 1, nb_classes=2, class_labels=['good', 'bad'])
    np.random.shuffle(alldata)
    add_samples_to_training_set(alldata, "good_file", 0)
    add_samples_to_training_set(alldata, "bad_file", 1)

    log.warning('Preparing data...')
    tstdata_temp, trndata_temp = alldata.splitWithProportion(0.3)

    tstdata = ClassificationDataSet(len(dictionary), 1, nb_classes=2)
    for n in range(0, tstdata_temp.getLength()):
        tstdata.addSample(tstdata_temp.getSample(n)[0], tstdata_temp.getSample(n)[1])

    trndata = ClassificationDataSet(len(dictionary), 1, nb_classes=2)
    for n in range(0, trndata_temp.getLength()):
        trndata.addSample(trndata_temp.getSample(n)[0], trndata_temp.getSample(n)[1])

    for data in [trndata, tstdata]:
        data._convertToOneOfMany()
```

```
    tries = 10
    epochs = 2
    verbose = True
    fast = False
    bias = True
    x = []
    y = []
    previous_error = 100
    log_file_name = log_name
    for _ in range(tries):
        x.append(_)
        log.warning('Constructing NeuralNetwork...')
        print(trndata.indim)
        try_fnn = buildNetwork(trndata.indim, trndata.indim * 2, trndata.outdim, hiddenclass=SigmoidLayer,
                               outclass=SoftmaxLayer, bias=bias, fast=fast)

        log.warning('Training NeuralNetwork...')
        trainer = BackpropTrainer(try_fnn, dataset=trndata, momentum=0.1, verbose=verbose, weightdecay=0.01)
        trainer.trainEpochs(epochs)

        log.warning('Computing train and test errors...')
        trnresult = percentError(trainer.testOnClassData(dataset=trndata), trndata['class'])
        tstresult = percentError(trainer.testOnClassData(dataset=tstdata), tstdata['class'])
        print("epoch: %4d" % trainer.totalepochs,
              "  train error: %5.2f%%" % trnresult,
              "  test error: %5.2f%%" % tstresult)
        if tstresult < previous_error:
            fnn = try_fnn
            previous_error = tstresult
    NetworkWriter.writeToFile(fnn, 'nn.xml')

    log.warning('Activating NeuralNetwork...')
    nginx_log = ClassificationDataSet(len(dictionary), 1, nb_classes=2)
    add_samples_to_training_set(nginx_log, log_file_name, 0)
    nginx_log._convertToOneOfMany()  # this is still needed to make the fnn feel comfy

    out = fnn.activateOnDataset(nginx_log)
    out = out.argmax(axis=1)  # the highest output activation gives the class

    with open(log_file_name) as log_file:
        cnt = 0
        for line in log_file:
            try:
                entry = LogEntry(*nginx_log_re.match(line).groups())
                if out[cnt]:
                    print("ATACK!!!:  "),
                else:
                    print("NOT ATACK: "),
                print("{0}".format(entry))
                cnt += 1
            except Exception:
                log.error('Failed to parse line: {0}'.format(line), exc_info=True)


if __name__ == '__main__':
    with Pool(5) as p:
        p.map(main_prog, 'log_file')
```

ВІДОМІСТЬ АТЕСТАЦІЙНОЇ РОБОТИ

Позначення	Найменування	Дод. відомості
	Текстові документи	
1	Пояснювальна записка	63 с.
2	Презентаційний матеріал	32 с.
	Інші документи	
3	Роздруківки програм	6 с.
4	Рецензія	2 с.
5	Відгук керівника	1 с.

					Методи виявлення DDoS-атак, засновані на застосуванні машинного навчання			
Змін	**Арк.**	**Номер докум.**	**Підп.**	**Дата**				
Розроб.		**Безіна К.Ю.**			**(Тема роботи) Відомість атестаційної роботи**		**Аркуш**	**Аркушів**
Перевір.		**Кіріченко Л.О.**						
Н. контр.		**Сидоров М.В.**				**ХНУРЕ кафедра ПМ**		
Затв.		**Тевяшев А.Д.**						