Санкт-Петербургский государственный политехнический университет
Институт машиностроения, материалов и транспорта
Кафедра «Мехатроника и Роботостроение» при ЦНИИ РТК

О Т Ч Е Т П О К У Р С О В О Й Р А Б О Т Е

Мониторинг отказов системы по ряду параметров на основе kNeighbours Regressor

по дисциплине «Математические методы интеллектуальных технологий»

Выполнил
студент гр.3341506/90401 <*подпись*> М.А.Борискин

Проверил
<*подпись*> А.В.Бахшиев

«25» декабря 2019 г.

Санкт-Петербург
2019

1. Создание набора данных (dataset).

Производители наиболее востребованных в настоящий момент квадрокоптеров – компания DJI – позволяет рядовому пользователю разрабатывать собственные мобильные приложения для платформ Android и iOS. Для этого реализован open-source DJI SDK (Software Development Kit).

В ходе разработки приложения пользователем, DJI SDK позволяет отслеживать критические ошибки, которые приводят к отказу приложения путем аварийного выхода из него и вывода на пользовательский интерфейс сообщения “Данное приложение не отвечает”.

Такие критические отказы в работе системы возникают из-за неверной логики разработки. Оптимизация такой логики и устранение возникающих ошибок является основой процесса разработки.

Технологические стек, используемый в ходе разработки приложения для квадрокоптеров DJI на Android ОС, следующий: Java 8 (Android-Studio), DJI SDK v4.3.0-v4.9.1, Gradle.

В Java существует система отслеживания возникающих критических ошибок. Они делятся на Exception отказы и на Error отказы.

Отказ – один из основных терминов теории надежности, означающий нарушение работоспособности объекта, при котором система или элемент перестает выполнять целиком или частично свои функции, иначе – сбой в работе устройства, системы.

Exception в Java – непредвиденное событие, которое произошло в ходе выполнения программы, то есть в run-time, которое повлекло неопределенное поведение программы. Error в Java – серьезная проблема, которую приложение не должно было поймать, возникшая также в ходе выполнения программы. Видно, что оба данных определения подпадают под термин отказ системы из теории наджености.

Обычно, такие проблемы не отслеживаются в ходе прямой работы приложения. Распространен подход отслеживания таких проблем с помощью unit-тестирования. Однако в случае приложений такого рода, как рассматриваемое в данной работе, то есть когда помимо Android-устройства также используются дополнительные девайсы, как пульт дистанционного управления и сам квадркоптер, подключенные к Android-устройству, корректное тестирование требует высокой квалификации и исследовательской работы разработчика, так как DJI SDK не предоставляет такой возможности по умолчанию. Однако тестирование работы разрабатываемого приложения зачастую необходимо производить при работе квадрокоптера в режиме полета.

В таком случае критические ошибки выводятся в реализуемый в Android ОС по умолчанию StackTrace в домашнюю директорию устройства. Если что-то в ходе работы приложения идет не так, то Java Virtual Machine начинает искать обработчики ошибок, если таких обработчиков нет, то исключение переходит по стеку выполнения вверх. Приложение аварийно прекращает работу. И в качестве автоматизированного логирования ошибки – в домашнюю директорию выводится наименование ошибки.

Подобный функционал констатации ошибок параллельно реализуется также средствами DJI SDK. В домашней директории устройства по относительному пути ./DJI/***/LOG/CRASH выводятся .txt-файлы с наименованием ошибки и вывод системной информации о процессе из /*proc*/PID/status, где PID – это идентификационный номер процесса в системе (рисунок 1).

```
=======Thread info======Crash name:java.lang.ArrayIndexOutOfBoundsException: length=12; index=-1
 Cause is:null
 Thread name is:dji_background_thread 422
 Thread count is:832
 Fd count is:266

Name:   com.rtc.drone
State:  S (sleeping)
Tgid:   5991
Pid:    5991
PPid:   209
TracerPid:      0
Uid:    10200   10200   10200   10200
Gid:    10200   10200   10200   10200
FDSize: 2048
Groups: 1015 1028 3001 3002 3003 9997 50200
VmPeak:  2438712 kB
VmSize:  2438712 kB
VmLck:        64 kB
VmPin:         0 kB
VmHWM:    273884 kB
VmRSS:    254076 kB
VmData:  1057936 kB
VmStk:      8192 kB
VmExe:        12 kB
VmLib:     97564 kB
VmPTE:      2512 kB
VmSwap:      852 kB
Threads:        847
SigQ:   1/13636
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000001204
SigIgn: 0000000000000000
SigCgt: 00000002000094f8
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 0000000000000000
Seccomp:        0
Cpus_allowed:   f
Cpus_allowed_list:      0-3
voluntary_ctxt_switches:        33673
nonvoluntary_ctxt_switches:     14770
```

Рисунок 1 – Пример файла, как записывается информация об ошибках средствами DJI SDK

Таким образом, в ходе работы было принято решение средствами машинного обучения отслеживать возникновение непредвиденных ошибок в ходе работы данного конкретного приложения для квадрокоптеров DJI как раз по ряду данных параметров из /*proc*/PID/status, которые предствлены на рисунке выше.

Эти параметры характиризуют работу процесса с файловой системой в Linux ядре (рисунок 2).

```
Name                        filename of the executable
State                       state (R is running, S is sleeping, D is sleeping
                            in an uninterruptible wait, Z is zombie,
                            T is traced or stopped)
Tgid                        thread group ID
Ngid                        NUMA group ID (0 if none)
Pid                         process id
PPid                        process id of the parent process
TracerPid                   PID of process tracing this process (0 if not)
Uid                         Real, effective, saved set, and  file system UIDs
Gid                         Real, effective, saved set, and  file system GIDs
FDSize                      number of file descriptor slots currently allocated
Groups                      supplementary group list
VmPeak                      peak virtual memory size
VmSize                      total program size
VmLck                       locked memory size
VmPin                       pinned memory size
VmHWM                       peak resident set size ("high water mark")
VmRSS                       size of memory portions. It contains the three
                            following parts (VmRSS = RssAnon + RssFile + RssShmem)
VmData                      size of private data segments
VmStk                       size of stack segments
VmExe                       size of text segment
VmLib                       size of shared library code
VmPTE                       size of page table entries
VmSwap                      amount of swap used by anonymous private data
                            (shmem swap usage is not included)
Threads                     number of threads
SigQ                        number of signals queued/max. number for queue
SigPnd                      bitmap of pending signals for the thread
ShdPnd                      bitmap of shared pending signals for the process
SigBlk                      bitmap of blocked signals
SigIgn                      bitmap of ignored signals
SigCgt                      bitmap of caught signals
CapInh                      bitmap of inheritable capabilities
CapPrm                      bitmap of permitted capabilities
CapEff                      bitmap of effective capabilities
CapBnd                      bitmap of capabilities bounding set
CapAmb                      bitmap of ambient capabilities
Seccomp                     seccomp mode, like prctl(PR_GET_SECCOMP, ...)
Cpus_allowed                mask of CPUs on which this process may run
Cpus_allowed_list           Same as previous, but in "list format"
voluntary_ctxt_switches     number of voluntary context switches
nonvoluntary_ctxt_switches  number of non voluntary context switches
```

Рисунок 2 – Выписка из документации по файловой системе Linux-ядра

Также в набор данных необходимо добавить не только характеристики ошибок по данным параметрам, но также и характеристики данного процесса для его работы в норме, то есть: нагенерировать данных параметров в моменты времени, когда программа приложения выполняется без ошибок.

На момент начала реализации данной курсовой работы – в папке с логированием ошибок средствами DJI SDK: 5632 валидных файла (в ходе работы средствами Python3.6 были отсеяны не валидные файлы: пустые; содержащие более одного наименования ошибки; содержащие только наименование ошибки без вывода параметров процесса).

В качетсве хост-машины (далее – система мониторинга) использовался персональный компьютер MSI GP72 с ОС Ubuntu Budgie 18.04.

Доступ к Android-устройству осуществляется с помощью утилиты командной строки – Android Device Bridge (*adb*). Работает на хосте отладки (система мониторига в данном случае). Обмен данным возможен между хостом и запущенным эмулятором Android-девайса или реальным устройством подключенным посредством USB или WiFi соединения. Работает данная утилита по технологии клиент-сервер и включает в себя следующие три компоненты:

- Клиент, работает на хосте отладки, то есть напрямую – система мониторинга.
- Сервер, запущен как фоновый процесс на хосте разработки. Управляет обменом данными между клиентом и демоном, работающим на Android-устройстве.
- Демон (служба), запущена как фоновый процесс на работающем Andorid-устройстве.

Работает данная утилита как прослушивание локального TCP порта 5037 на localhost (рисунок 3).

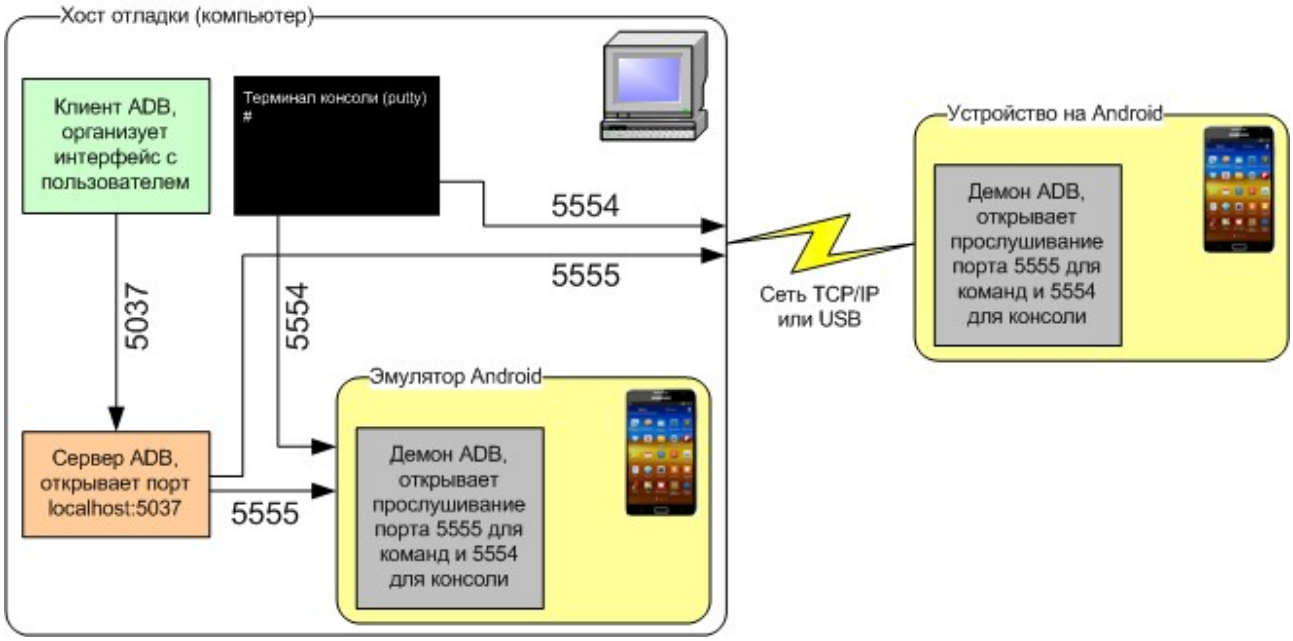

Рисунок 3 – Схема работы утилиты *adb*

Для работы с данной утилитой существует множество команд, отвечающих философии клиент-сервер. Таким образом, помимо прочих также существует команда ***adb pull -from -to***, где -from – это абсолютный адрес откуда на Android-устройстве стягивать файл, -to – абсолютный адрес куда стягивать этот файл на хост машину.

В случае данной работы не получится просто так стянуть файл из /*proc*/PID/status, так как это системный файл, и даже из под root-пользователя не получается стянуть напрямую данный файл на хост машину.

Поэтому, обойти данную проблему в текущей работе было решено следующим образом: в Android-устройство помещается SD-накопитель. С помощью bash скрипта (рисунок 4) из под root-пользователя Android-устройства

производится вывод всего содержимого файла /*proc*/PID/status для нашего работающего приложения, в .txt-файл на SD-накопителе. А затем уже с SD-накопителя командой ***adb pull*** файл стягивается в определенную директорию на хост машину, то есть на систему мониторинга. И используется далее.

```
root@shieldtablet:/ # while true; do echo -n "this norm was writen like "; date; cat /proc/4569/status > /
storage/sdcard1/1223/state.txt; sleep 1;  done
```

```
makary@pafeast:~$ while true; do adb pull /storage/sdcard1/1223/state.txt /home/
makary/Documents/ml_1sem_5kurs/k/src/monitoring/state_1.txt; sleep 1; done
```

Рисунок 4 – bash-скрипты для передачи параметров процесса на систему мониторинга

Подключение квадрокоптера и ПДУ к Andorid-устройству и системе мониторинга было осуществлено следующим образом, как показано на следующем рисунке 5 ниже.

Рисунок 5 – Структурная схема работы системы мониторинга

Таким образом, сперва для обучения регрессии было нагенерировано примерно 4732 .txt-файла с параметрами, характеризующими нормальную работу приложения. Статус процесса выводился каждую секунду, для вывода столько файлов потребовалось три полноценных разряда батареи квадрокоптера.

На языке программирования python3.6 был написал парсер полученных .txt-файлов в один удобочитаемый .csv документ.

Принцип работы парсера следующий: осуществляется проход по всем строкам каждого файла. На основании него составляются 39 различных pandas

dataFrame, содержащих по одному столбцу для каждого параметра набора данных. Затем все столбцы конкатенируются в один dataFrame и сохраняются как .csv-файл средствами pandas. Пример работы основных функций, помимо описанной выше построчной записи и конкатенации, приведен на рисунке 6.

Такой подход к парсингу для данного набора данных занял 7 минут 29 секунд 89 миллисекунд. С другой стороны подход с парсингом в один dataFrame занял 6 часов 24 минут 39 секунд 16 миллисекунд. Данное время измерялось внутренним средствами python 3.6 (функцией time()).

В результате, с помощью утилиты *adb* и bash-скриптов удалось собрать набор данных для обучения модели регрессии: 39 столбцов на 10243 строки.

```
def set_12_VmSize(line_n, data, n):
    VmSize = getParam(line_n, "VmSize")
    if VmSize is not None:
        print(n, VmSize)
        data = data.append({'VmSize': VmSize}, ignore_index=True)
        return data
    return data
```

```
def getParam(line_n, parameter):
    if parameter == "java.lang.":
        if line_n[0:10] == "java.lang.":
            words = line_n.replace('.', '').split()
            error = words[0]
            return error[8:len(error) - 1]
        if line_n[0:len("android.view.WindowManager$")] == "android.view.WindowManager$":
            words = line_n.replace('.', '').split()
            error = words[0]
            return error[len("android.view.WindowManager$")-2:len(error) - 1]
        if line_n[0:len("android.os.")] == "android.os.":
            words = line_n.replace('.', '').split()
            error = words[0]
            return error[len("android.os.") - 2:len(error)]
        if line_n[0:len("android.view.ViewRootImpl$")] == "android.view.ViewRootImpl$":
            words = line_n.replace('.', '').split()
            error = words[0]
            return error[len("android.view.ViewRootImpl$") - 2:len(error) - 1]

    elif line_n[0:len(parameter)] == parameter:
        words = line_n.split()
        if parameter == "Uid" or parameter == "Gid":
            return words[1]+" "+words[2]+" "+words[3]+" "+words[4]
        elif parameter == "Groups":
            return words[1]+" "+words[2]+" "+words[3]+" "+words[4]+\
                   " "+words[5]+" "+words[6]+" "+words[7]
        else:
            return words[1]

    else:
        return None
```

Рисунок 6 – Пример основных функций для парсинга файлов в данной курсовой работе

2. Анализ набора данных

Для начала выведем список версий используемых в работе python3.6 библиотек (рисунок 7).

```
Numpy version: 1.17.4
Pandas version: 0.25.3
Matplotlib version: 3.1.2
Seaborn version: 0.9.0
Geopandas version: 0.6.2
Plotly version: 4.3.0
Scikit-Learn version: 0.21.3
```

Рисунок 7 – Список версий используемых в работе библиотек python3.6

Выведем длину набора данных. Почистим набор данных, убрав не изменяющиеся и не числовые значения. А также выведем после этого первые пять строк (рисунок 8).

```
print("Length of CRASHES DataFrame =", len(data['error_name']))
```

Результат:

```
Length of CRASHES DataFrame = 10244
```

```
data.drop(['State', 'TracerPid', 'FDSize', 'VmPin', 'VmStk',
           'VmExe', 'SigPnd', 'ShdPnd', 'SigBlk', 'SigIgn',
           'SigCgt', 'CapInh', 'CapPrm', 'CapEff', 'CapBnd',
           'Seccomp', 'Cpus_allowed', 'Cpus_allowed_list', 'n'], axis = 1, inplace = True)
```

```
print(data.head())
```

Результат:

	error_name	Tgid	...	nonvoluntary_ctxt_switches	crash
0	NullPointerException	6220	...	18680	1
1	ArrayIndexOutOfBoundsException	5991	...	14770	2
2	OutOfMemoryError	2469	...	38149	3
3	OutOfMemoryError	9548	...	31378	3
4	NullPointerException	13124	...	649	1

Рисунок 8 – Трансформация и чистка набора данных

Итоговая размерность набора данных следующая: 20 столбцов на 10244 строк.

В качестве примера выведем первую строку набора данных, раскиданную по столбцам (рисунок 9).

```
print(data.loc[0])
```

Результат:

```
error_name                                NullPointerException
Tgid                                                      6220
Pid                                                       6220
PPid                                                       218
Uid                                   10200 10200 10200 10200
Gid                                   10200 10200 10200 10200
Groups                  1015 1028 3001 3002 3003 9997 50200
VmPeak                                                 2701464
VmSize                                                 2701456
VmLck                                                       64
VmHWM                                                  740056
VmRSS                                                  495012
VmData                                                1338128
VmLib                                                   79964
VmPTE                                                    3068
VmSwap                                                   3244
Threads                                                  1087
SigQ                                                 1/13636
voluntary_ctxt_switches                                  8392
nonvoluntary_ctxt_switches                              18680
crash                                                       1
Name: 0, dtype: object
```

Рисунок 9 – Первая строка набора данных, раскиданная по столбцам

Также выведем полный список уникальных имен всех отказов, включая имя нормального поведения (рисунок 10).

```
names = sorted(set(data['error_name']))
print(names)
```

```
['ArrayIndexOutOfBoundsException', 'BadTokenException', 'CalledFrom
WrongThreadException', 'ClassNotFoundException', 'IllegalMonitorSta
teExceptio', 'IllegalStateException', 'InternalError', 'NetworkOnMa
inThreadException', 'NoClassDefFoundError', 'NormalBehaviou', 'Null
PointerException', 'OutOfMemoryError', 'RuntimeException']
```

Рисунко 10 – Список уникальных имен всех отказов системы, включая нормальное поведение

Также для визуального понимания выведем общее количество вхождений каждого отказа в наш набор данных (рисунок 11).

```
plt.figure(figsize=(40, 20))
sns.countplot(data['error_name'], label="Count")
plt.savefig("density.png")
```

Результат:

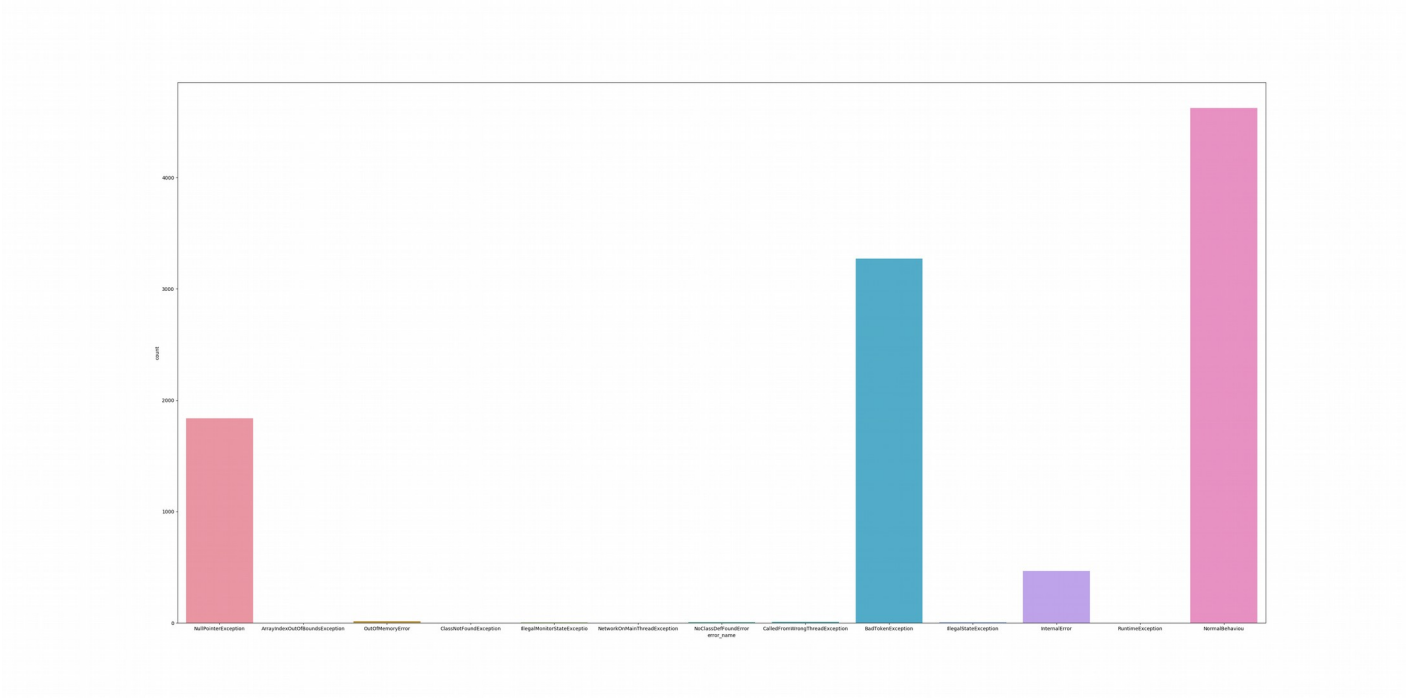

Рисунок 11 – Количество вхождений каждого уникального наименования отказа, включая нормальное поведение (самый правый столбец)

Как видно из данной гистограммы, наибольшее количество вхождений имеют отказы: NullPointerException, BadTokenException, InternalError. Такой набор данных не идеален, так как количество других ошибок мало, что не дает уверенности в верном определеннии их возникновения при мониторинге. В дальнейшем необходимо расширить данный набор данных, уравняв число вхождений в него для всех отказов системы.

Также на данный момент в данный набор данных входят только те отказы, что были встречены в период разработки приложения. На самом деле возможных исключений и ошибок больше. Также в ходе дальнейшей работы необходимо добавить в обработку их, и тем самым скорректировать набор данных.

Также построим матрицу корреляции используемых параметров следующим образом (рисунок 12).

```
data.corr().style.background_gradient(cmap='coolwarm')
plt.figure(figsize=(12, 12))
ax = sns.heatmap(data.corr(), annot=True)
plt.savefig("corr.png")
```

Результат:

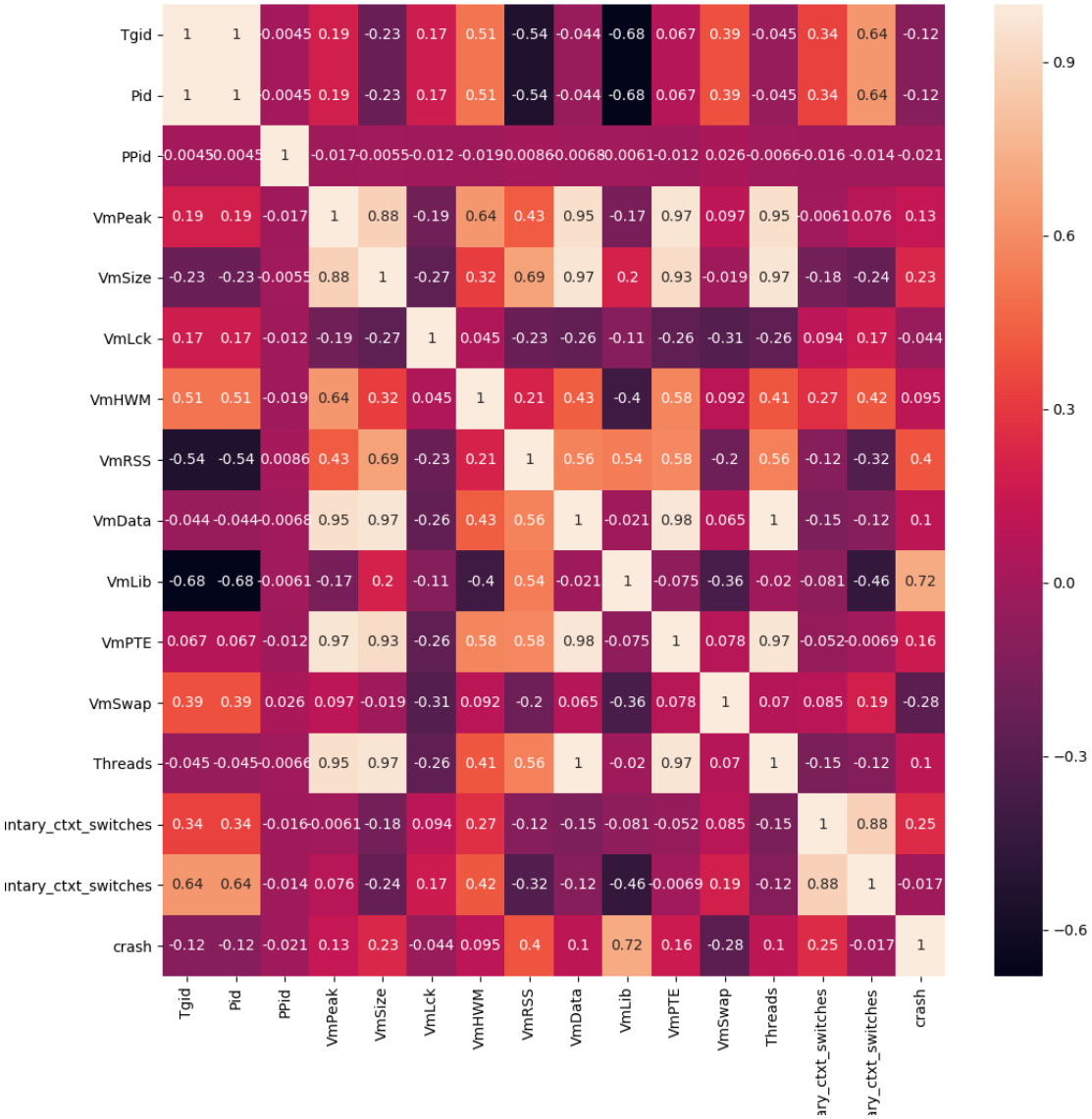

Рисунок 12 – Матрица корреляции используемых параметров

Из данных результатов видно, что некоторые параметры коррелируют с конкретными отказами системы и между собой, чего и следовало ожидать от характеристик одного процесса. Таким образом, использование данного набора данных в условиях данной задачи оправдано.

Также построим гистограммы всех используемых признаков (рисунок 13).

```
data.hist(figsize=(15, 8), layout=(3, 6))
plt.savefig("hist.png")
```

Результат:

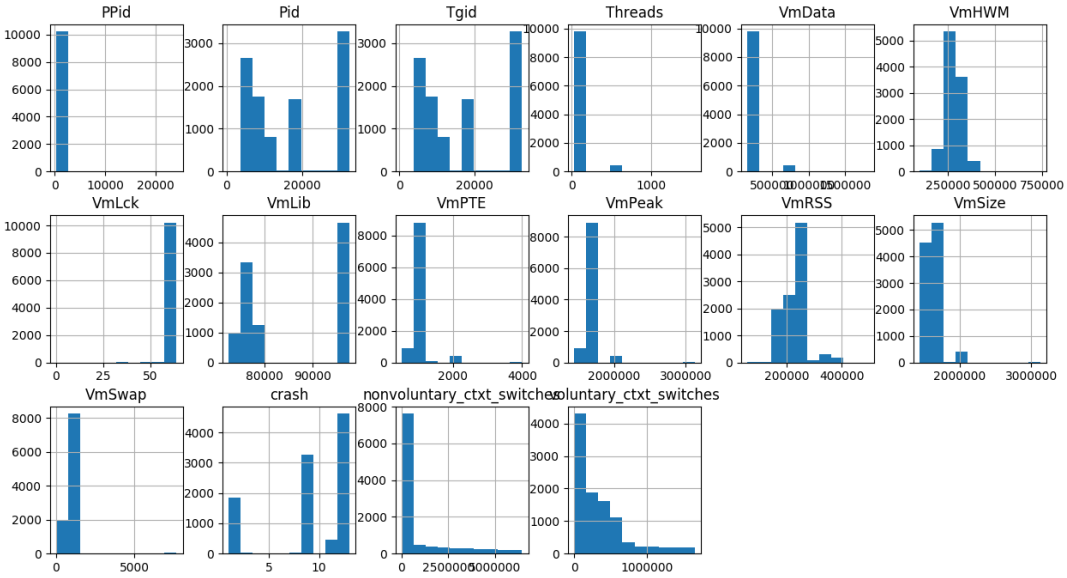

Рисунок 13 – Гистограммы всех используемых признаков

Из данных гистограмм можно видеть, что значения одних и тех же парамтеров розняца в различных случаях, что также позволяет судить о возможности использования данного набора данных в условиях нашей задачи мониторинга отказов системы по ряду параметров.

3. Модель

После создания набора данных для обучения путем исследования и последующего парсинга, а также после проведения анализа получевшегося набора данных необходимо обучить используемую модель и применить результаты для работы системы мониторинга.

Сперва классифицируем ошибки на основе решающих деревьев, используя классификатор *DecisionTreeClassifier* из библиотеки *scikit-learn* (рисунок 14).

Будем использовать энтропийный критерий построения дерева поиска.

```
x_train, x_test, y_train, y_test = train_test_split(df_knn[['VmPeak',
                                                            'VmSize',
                                                            'VmLck',
                                                            'VmHWM',
                                                            'VmRSS',
                                                            'VmData',
                                                            'VmLib',
                                                            'VmPTE',
                                                            'VmSwap',
                                                            'Threads',
                                                            'voluntary_ctxt_switches',
                                                            'nonvoluntary_ctxt_switches']], df_knn['crash'],
                                                    test_size=0.2,
                                                    random_state=42)
```

```
crit = 'entropy'

t1 = time()
clf_tree = DecisionTreeClassifier(criterion = crit, max_depth = 19, random_state = 20, presort = True)

clf_tree.fit(X = x_train, y = y_train)

err_train = round(np.mean(y_train != clf_tree.predict(x_train)) * 100, 4)
err_test = round(np.mean(y_test != clf_tree.predict(x_test)) * 100, 4)

t = -(t1 - time())

print("Глубина дерева: {}, ошибка на обучающей: {}, ошибка на тестовой: {}, время {}"
                                                    .format(clf_tree.get_depth(), err_train, err_test, t))
```

Результат:

```
Глубина дерева: 13, ошибка на обучающей: 0.0, ошибка на тестовой: 0.6345, время 0.0419461727142334
```

Рисунок 14 – Результат работы классификатора на основе решающих деревьев

Как видно, классификатору удалось успешно классифицировать отказы и нормальное поведение системы. Ошибка классификации: 0.6345%.

Из такого результата можно судить о том, что характеризующие процесс в Linux ядре параметры подходят для вычисления по ним конкретного отказа или же нормального поведения системы.

Далее обучим на нашем наборе данных kNeighbours Regressor и посмотрим результаты его предсказаний на тестовой выборке (рисунок 15).

```
neighbours = [2, 4, 5, 6, 7, 8, 10, 12, 15, 19]
for i in neighbours:
    knn = KNeighborsRegressor(n_neighbors=i,
                              weights='uniform',
                              algorithm='auto',
                              leaf_size=30, p=2,
                              metric='minkowski',
                              metric_params=None,
                              n_jobs=None)
    knn.fit(x_train, y_train)

    predictions_test = knn.predict(x_test)
    predictions_test = pd.DataFrame({"error_name": predictions_test})

    mae = mean_absolute_error(y_test, predictions_test)

    print("Для KNeighborsRegressor:\n")
    print("Для n_neighbours = ", i)
    print('mean_squared_log_error:\t%.5f' % mean_squared_log_error(y_test, predictions_test))
    print('mean_absolute_error:\t(пункты)%.4f' % mean_absolute_error(y_test, predictions_test))
    print("Median Absolute Error: " + str(round(median_absolute_error(predictions_test, y_test), 2)))
    RMSE = round(sqrt(mean_squared_error(predictions_test, y_test)), 2)
    print("Root mean_squared_error: " + str(RMSE))
    print("\n")
    print("\n")
```

Результат:

```
Для n_neighbours =  2
mean_squared_log_error: 0.01468
mean_absolute_error:    (пункты)0.0649
Median Absolute Error: 0.0
Root mean_squared_error: 0.64

Для n_neighbours =  4
mean_squared_log_error: 0.01384
mean_absolute_error:    (пункты)0.0667
Median Absolute Error: 0.0
Root mean_squared_error: 0.63

Для n_neighbours =  5
mean_squared_log_error: 0.01400
mean_absolute_error:    (пункты)0.0663
Median Absolute Error: 0.0
Root mean_squared_error: 0.62

Для n_neighbours =  6
mean_squared_log_error: 0.01372
mean_absolute_error:    (пункты)0.0665
Median Absolute Error: 0.0
Root mean_squared_error: 0.62

Для n_neighbours =  7
mean_squared_log_error: 0.01465
mean_absolute_error:    (пункты)0.0701
Median Absolute Error: 0.0
Root mean_squared_error: 0.64

Для n_neighbours =  8
mean_squared_log_error: 0.01558
mean_absolute_error:    (пункты)0.0745
Median Absolute Error: 0.0
Root mean_squared_error: 0.66

Для n_neighbours =  10
mean_squared_log_error: 0.01583
mean_absolute_error:    (пункты)0.0761
Median Absolute Error: 0.0
Root mean_squared_error: 0.67

Для n_neighbours =  12
mean_squared_log_error: 0.01691
mean_absolute_error:    (пункты)0.0774
Median Absolute Error: 0.0
Root mean_squared_error: 0.69

Для n_neighbours =  15
mean_squared_log_error: 0.01716
mean_absolute_error:    (пункты)0.0806
Median Absolute Error: 0.0
Root mean_squared_error: 0.7

Для KNeighborsRegressor:

Для n_neighbours =  19
mean_squared_log_error: 0.01883
mean_absolute_error:    (пункты)0.0874
Median Absolute Error: 0.0
Root mean_squared_error: 0.73
```

Рисунок 15 – Результат работы kNeighbours Regressor для разного количества neighbours

Из полученных результатов было выбрано n_neighbours = 6 для использования в работе.

Выбор вывода ошибок для характеристики регрессии, а также выбор по итоговым значениям ошибок был сделан на основании теоретического материала, представленного в главе “Метрики качества”, размещенных в https://alexanderdyakonov.files.wordpress.com/2018/10/book_08_metrics_12_blog1.pdf .

4. Система мониторинга

После обучения модели регрессии необходимо переходить к описанию самой системы мониторинга.

Она была реализована на языке программирования python3.6.

Принцип работы системы мониторинга был представлен на рисунке 5 ранее. Для более четкого понимания, необходимо дополнить данную структурную схему до той, что показана на рисунке 16.

Рисунок 16 – Структурная схема работы системы мониторинга

Выбор количества обрабатываемых файлов – 8 – был сделан из расчета – не много, не мало. Исследований о влиянии количества на результат (качество и время) не проводилось.

Вывод работы системы мониторинга осуществляется как показано на рисунках 17 и 18.

В ходе экспериментальных исследований было выявлено успешное определение системой мониторинга нормального поведения в случае разного вида работы с приложением (рисунок 17). А также верного определения возникающих ошибок в случае запуска в работу программы с заранее известным сбоем (OutOfMemoryError и последующим NullPointerException.

Сообщения обрабатываются и выводятся с частотой 0.25 секунды, так как такой формат наиболее воспринимаем человеком для прочтения в случае возникновения ошибки и осуществления реакции на нее.

Также параллельно осуществляет вывод текущего набора из 8 файлов, чтобы пользователь мог следить, что параметры в них меняются (это говорит о том, что система на планшете не зависла и bash-скрипт продолжает выполняться и обновлять данные.

```
   n      error_name  ... nonvoluntary_ctxt_switches  crash
0  1  NormalBehaviou  ...                       19473     13
1  2  NormalBehaviou  ...                       18771     13
2  3  NormalBehaviou  ...                       18860     13
3  4  NormalBehaviou  ...                       18967     13
4  5  NormalBehaviou  ...                       19063     13
5  6  NormalBehaviou  ...                       19204     13
6  7  NormalBehaviou  ...                       19267     13
7  8  NormalBehaviou  ...                       19401     13

[8 rows x 40 columns]
/**************************************************

ALL RIGHT:  NormalBehaviour probability: 92.34585976899301 %

**************************************************/
   n      error_name  ... nonvoluntary_ctxt_switches  crash
0  1  NormalBehaviou  ...                       19473     13
1  2  NormalBehaviou  ...                       19543     13
2  3  NormalBehaviou  ...                       18860     13
3  4  NormalBehaviou  ...                       18967     13
4  5  NormalBehaviou  ...                       19063     13
5  6  NormalBehaviou  ...                       19204     13
6  7  NormalBehaviou  ...                       19267     13
7  8  NormalBehaviou  ...                       19401     13

[8 rows x 40 columns]
/**************************************************

ALL RIGHT:  NormalBehaviour probability: 92.34585976899301 %

**************************************************/
```

Рисунок 17 – Вывод системы мониторинга при работе в нормальном режиме

```
[8 rows x 40 columns]
/*************************************************

WARNING:  OutOfMemoryError probability: 94.4525784935741 %

*************************************************/
   n      error_name  ... nonvoluntary_ctxt_switches  crash
0  1  NormalBehaviou  ...                      21080     13
1  2  NormalBehaviou  ...                      21080     13
2  3  NormalBehaviou  ...                      21080     13
3  4  NormalBehaviou  ...                      21080     13
4  5  NormalBehaviou  ...                      21080     13
5  6  NormalBehaviou  ...                      21080     13
6  7  NormalBehaviou  ...                      21080     13
7  8  NormalBehaviou  ...                      21080     13

[8 rows x 40 columns]
/*************************************************

WARNING:  OutOfMemoryError probability: 94.4525784935741 %

*************************************************/
   n      error_name  ... nonvoluntary_ctxt_switches  crash
0  1  NormalBehaviou  ...                      21080     13
1  2  NormalBehaviou  ...                      21080     13
2  3  NormalBehaviou  ...                      21080     13
3  4  NormalBehaviou  ...                      21080     13
4  5  NormalBehaviou  ...                      21080     13
5  6  NormalBehaviou  ...                      21080     13
6  7  NormalBehaviou  ...                      21080     13
7  8  NormalBehaviou  ...                      21080     13
```

```
/*************************************************

WARNING:  OutOfMemoryError probability: 94.4525784935741 %

*************************************************/
   n      error_name  ... nonvoluntary_ctxt_switches  crash
0  1  NormalBehaviou  ...                    21080.0     13
1  2  NormalBehaviou  ...                    21080.0     13
2  3  NormalBehaviou  ...                    21080.0     13
3  4  NormalBehaviou  ...                    21080.0     13
4  5  NormalBehaviou  ...                        NaN     13
5  6  NormalBehaviou  ...                        NaN     13
6  7  NormalBehaviou  ...                        NaN     13
7  8  NormalBehaviou  ...                        NaN     13

[8 rows x 40 columns]
/*************************************************

WARNING:  NullPointerException  probability: 44.4525784935741 %

*************************************************/
   n      error_name  ... nonvoluntary_ctxt_switches  crash
0  1  NormalBehaviou  ...                    21080.0     13
1  2  NormalBehaviou  ...                    21080.0     13
2  3  NormalBehaviou  ...                    21080.0     13
3  4  NormalBehaviou  ...                        NaN     13
```

Рисунок 18 – Вывод системы мониторинга при запуске “нерабочей” версии

Расчет вероятности (probability в выводе на рисунках 17, 18) точного предсказания данного отказа или нормального поведения системы происходит из следующего принципа:

- Средняя абсолютная ошибка работы используемой модели на тестовой выборке во время обучения. В данном случае 6,65%.
- Количество входов данной ошибки в 8 проверенных файлов состояний системы.

То есть расчет идет таким образом: в случае, когда все восемь тестовых состояний определились, как OutOfMemoryError, то из 100% вычитается 6.65% и выводится результат. В случае, когда 4 из 8 определились как NullPointerException, а другие 4, как OutOfMemoryError, выводится результат предсказания по последнему на данный момент файлу, а вероятность высчитывается как 50% - 6.65%.

Такой подход к расчету вероятности сделан из удобства. В дальнейшем необходимо пересмотреть подход к определению вероятности. Необходимо использовать следующий подход: провести перерасчет ошибки по принципу распределения случайной величины, а провести исследование о точных характеристиках описываемых результаты работы регрессионной модели ошибок и также включить их в расчет.

Выводы по результатам работы:

В ходе данной курсовой работы была разработана и протестирована на небольшом количестве примеров функциональная система мониторинга работы Android-приложения для управления квадрокоптерами DJI.

Данная система может быть использована для отладки работы приложения на стадии его разработки, а также для использования как параллельно функционирующее программное обеспечение, поставляемое вместе с оригинальным программным продуктом.

Однако для полноценной работы данной системы мониторинга необходимо провести дополнительные исследования и добавить в используемый для обучения набор данных большее количество примеров для существующих исключений и ошибок, уровняв их. А также необходимо добавить отсутствующие исключения и ошибки, которые также могут быть встречены в ходе разработки, тестирования и эксплуатации данного программного продукта.

Используемая в текущей версии системы мониторинга регрессионная модель kNeighboursRegressor обучена на предсказание ошибок для конкретной модели планшета, версии Linux ядра, а также версии Android ОС.

Для создания универсальной модели необходимо добавить в систему мониторинга программу, которая будет тестировать текущий аппратный и программного уровни Android-устройство, вызывая и обрабатывая все возможные ошибки и исключения и составляя свой набор данных для обучения модели регрессии. Данный процесс будет занимать время, однако в итоге получится свой набор данных для конкретного девайса, который можно будет использовать в системе мониторинга. Также система должна иметь возможность обучаться параллельно работе устройства для лучшего предсказания работы система. Также она должна использовать мало ресурсов, работая параллельно и не обременяя функционал остальной системы.