Московский Государственный Технический Университет имени Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»


# Отчёт по домашнему заданию № 1 по дисциплине «Проектирование интеллектуальных систем»

Исполнитель:
Саврасов П.А.
Группа ИУ5-24М
«__» _____ 2021 г.

Преподаватель:
Терехов В.И.
_____
«__» _____ 2021 г.

Москва, 2021 г.

# 1. Задание

Подготовить собственный датасет для дальнейшего обучения нейронной сети.

# 2. Создание датасета

```
import re
import json
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Файл data.txt - запись необработанных сообщений IRC чата выполненный специальным клиентом на Qt C++. Сообщения в данном наборе записаны в следующем формате: < Message > Raw content < /Message >. На данном этапе требуется отфильтровать сообщения, при этом учитывая, что сообщения бывают разного типа: сообщения подключения (метка JOIN), сообщения отключения (метка PART), сррбщения управления (метки CLEARCHAT, CLEARMSG)и пользовательские сообщения (метка PRIVMSG). Так же присутствуют сообщения технического характера (метка PING), как напримекр сообщения поддержания соединения. В выборке необходимо оставить только сообщения пользователей и сообщения управления, так как на их основе потом будут определяться класс пользовательских сообщений.

```
with open('data.txt', 'r') as file:
    data = file.read()
data = data.split('</Message>')
print('Записей в исходной выборке: ', len(data))
controlMsgs = []
userMsgs = []
for message in data:
    if 'CLEARCHAT' in message:
        controlMsgs.append(message)
    if 'CLEARMSG' in message:
        controlMsgs.append(message)
    if 'PRIVMSG' in message:
        userMsgs.append(message)
print('Пользовательских сообщений: ', len(userMsgs))
print('Управляющих сообщений: ', len(controlMsgs))
```

```
Записей в исходной выборке:  12787
Пользовательских сообщений:  12591
Управляющих сообщений:  164
```

Рассмотрим формат пользовательского сообщения:

- **badge-info=;**
- **badges=;**
- **client-nonce=7d0d0ea72a1cc2e687a76c11708ace7d;**
- **display-name=username;** - имя пользователя в чате
- **emotes=;**
- **flags=0-2:P.5;**
- **id=7253befd-92c7-4927-8e2c-9106ccc365fa;** - идентификатор сообщения
- **mod=0;** - флаг модерации
- **room-id=71092938;** - идентификатор чата
- **subscriber=0;** - флаг подписчика
- **tmi-sent-ts=1624267964499;**
- **turbo=0;**
- **user-id=570813914;** - идентификатор пользователя
- **user-type= :username!username@username** - тип пользователя
- **PRIVMSG #channel_name : message content\n'** - PRIVMSG - метка пользовательского сообщения, #channel_name - назщвание канала, : message content\n - контекст сообщения в экранировании : и \n

Из этих полей необходимыми являются только: display-name, id, user-id, и текст сообщения

Теперь рассмотрим сообщение управляющего типа:

- **ban-duration=100;** - время отстранения от чата в секундах. Если отсутсвует, то отстранение без ограничения по времени
- **room-id=71092938;** - идентификатор чата
- **target-user-id=169393458;** - идентификатор отстраняемого пользователя
- **tmi-sent-ts=1624267964499;**
- **CLEARCHAT #channel_name : username\n'** - CLEARCHAT - метка уравляющего сообщения, #channel_name - назщвание канала, : username\n - имя отстраняемого пользователя в экранировании : и \n

Для разбора сообщений применим классы сообщений, конструкторы которых будут разбирать строку сообщения по полям.

```
class UserMessage:
    def __init__(self, datastr : str):
        datastr.replace('<Message>', '')
        fields = datastr.split(';')
        for field in fields:
            if field.startswith('display-name='):
                self.display_name = field.split('=')[1]

            if field.startswith('id='):
                self.msg_id = field.split('=')[1]

            if field.startswith('user-id='):
                self.user_id = int(field.split('=')[1])

            if field.startswith('tmi-sent-ts='):
                self.tmi_sent_ts = int(field.split('=')[1])

        tmp = datastr.split('PRIVMSG #')[1].split(':')[1:]
        for substr in tmp:
            self.content += substr
        self.content = self.content.replace('\n', '')

    def __str__(self):
        return  'name: ' + self.display_name + \
                '\nuser id: ' + str(self.user_id) + \
                '\nmsg id: ' + self.msg_id + \
                '\ntimestamp: ' + str(self.tmi_sent_ts) + \
                '\ncontent: ' + self.content

    def toDict(self):
        return {
            'message_id': self.msg_id,
            'user_id': self.user_id,
            'username': self.display_name,
            'timestamp': self.tmi_sent_ts,
            'target': self.msg_class,
            'content': self.content
        }

    display_name = ''
    msg_id = ''
    user_id = 0
    content = ''
    tmi_sent_ts = 0
    msg_class = 0
```

```
userMsgObjects = []

for msg in userMsgs:
    userMsgObjects.append(UserMessage(msg))
```

Повторим тоже самое для классов управления:

```
class CtrlMessage:
    def __init__(self, datastr : str):
        datastr.replace('<Message>', '')
        fields = datastr.split(';')
        for field in fields:
            if field.startswith('@ban_duration='):
                self.msg_id = int(field.split('=')[1])

            if field.startswith('target-user-id='):
                self.target_user_id = int(field.split('=')[1])

            if field.startswith('tmi-sent-ts='):
                self.tmi_sent_ts = int(field.split('=')[1].split(' ')[0])

    def __str__(self):
        return  'ban duration: ' + str(self.ban_duration) + \
                '\ntarget user id: ' + str(self.target_user_id) + \
                '\ntimestamp: ' + str(self.tmi_sent_ts) + '\n'

    ban_duration = 0
    target_user_id = 0
    tmi_sent_ts = 0
```

```
ctrlMsgObjects = []

for msg in controlMsgs:
    ctrlMsgObjects.append(CtrlMessage(msg))
```

Теперь на основе сообщений управления необходимо найти те сообщения, которые привели к отстранениюю, то есть найти сообщения с нарушениями. Для этого необходимо связать идентификатор пользователя в обоих клссах и среди его сообщений найти то, что было последним по времени до врмемнной метки в управляющем сообщении

```
for ctrMsg in ctrlMsgObjects:
    tmp = None
    for usrMsg in userMsgObjects:
        if(ctrMsg.target_user_id == usrMsg.user_id):
            #if(ctrMsg.tmi_sent_ts >= usrMsg.tmi_sent_ts):
                tmp = usrMsg
    if tmp != None:
        tmp.msg_class = 1
```

Однако, этого недостаточно. В наборе управляющих сообщений было всего 164 сообщения, что означает что сообщений с нарушениями всего 164 из 12591, что составляет всего лишь 1,3%. Это связано с тем, что в необработанных данных нет информации от ботов (автоматических систем управления чатами), удаляющих сообщения с циклическим повторением символов. Для поиска таких сообщений воспользуемся функцией поиска циклических повторений. Чувствительность sensitivity устанавливает, какая минимальная часть сообщения должна повториться, чтобы сообщение можно было считать циклически повторяющимся. По умолчанию значение установлено на 05, то есть половину длины.

```
def repeatsDetector(content : str, sensitivity = 0.5):
    strLen = len(content)
    repeatMatrix = np.zeros((strLen, strLen))

    for i in range(strLen):
        for j in range(strLen):
            if(content[i] == content[j]):
                repeatMatrix[i][j] = 1

    repeatsSumm = np.zeros(strLen);
    for i in range(strLen):
        for j in range(strLen):
            if(repeatMatrix[j][(j + i) % strLen]):
                repeatsSumm[i] += 1

    repeatsCount = 0
    for i in range(strLen):
        if(repeatsSumm[i] >= strLen * sensitivity):
            repeatsCount += 1

    return repeatsCount
```

Проверим работу функции на примере сочитаний символов:

- abcdefg - в строке присутствует повторение 1 раз
- abcdabc - в строке присутствует повторение 1 раз
- abcabcabc - в строке присутствует повторение 3 раза
- abcabcab - в строке присутствует повторение 2 раза
- aaaaaaaaaa - в строке присутствует повторение 10 раз

```
testVars = ['abcdefg', 'abcdabc', 'abcabcabc', 'abcabcab', 'aaaaaaaaaa']
for testVar in testVars:
    print(testVar, repeatsDetector(testVar))
```

```
abcdefg 1
abcdabc 1
abcabcabc 3
abcabcab 3
aaaaaaaaaa 10
```

Теперь стоит подумать о других пороговых значениях. Сообщение можно не обрабатывать, если его длина менее 10 символов. Так же можно установить порог на число повторений. Пусть допустимое число повторений будет 3.

```
maxRepeats = 3
minLength = 10

for usrMsg in userMsgObjects:
    if(len(usrMsg.content) > 10):
        if(repeatsDetector(usrMsg.content) > 3):
            usrMsg.msg_class = 1
```

Теперь рассмотрим число сообщений с классом 1:

```
c1Count = 0
for usrMsg in userMsgObjects:
    if (usrMsg.msg_class == 1):
        c1Count += 1

print(c1Count)
```

```
1319
```

Теперь соотношение составляет 1319 из 12591, что уже хотя бы 11%. Теперь нужно подготовить сообщения для дальнейшего использования, записав их в файл в json формате.

```
outputDict = {'messages': []}
for usrMsg in userMsgObjects:
    outputDict['messages'].append(usrMsg.toDict())

with open('messages.json', 'w') as file:
    json.dump(outputDict, file, ensure_ascii=False)
```

Проверим корректность загрузив датасет через библиотеку pandas

```
with open('messages.json', 'r') as file:
    data = json.loads(file.read())

df = pd.json_normalize(data['messages'])
df.head()
```