Национальный исследовательский университет

“Высшая школа экономики”

Московский институт электроники и математики

“Прикладная математика”

Научно-исследовательский семинар

“Разработка аналитического сервиса с интерфейсом чат-бота”

Руководитель работы

доцент Яворский Р.Э.

Работу выполнил студент группы БПМ-171

Чернявский Владимир

Москва 2018

Введение

Blockchain - выстроенная по определённым правилам непрерывная последовательная цепочка блоков (связный список), содержащих информацию. Чаще всего копии цепочек блоков хранятся на множестве разных компьютеров независимо друг от друга[2]. Система *blockchain* нашла себе применение в сфере финансов - криптовалюты. Современные криптовалютные биржи, входящие в топ-10 по объему торгов, ежедневно зарабатывают на комиссионных сборах более $3 млн. В ноябре прошлого года была зафиксирована сумма в $23 млрд. по совокупным суточным торгам на биржах, торгующими криптовалютами. Блокчейн сейчас на пике популярности, ежедневно выходит огромное количество новостей из разных источников. Люди, которые хотят быть в курсе всех изменений, просто не могут себе позволить тратить несколько часов в день листая ленту. Таким образом растет потребность в агрегационных сервисах. Здесь появляются чат-боты.

Чат-бот - программа, которая может общаться с пользователями (имитируя поведение человека) для достижения какой-либо цели. Обычно это делается через платформу обмена сообщениями. Популярность таких ботов обусловлена тем, что они заменяют внешние сайты и приложения, позволяя пользователю не выходить из комфортного интерфейса мессенджера. Уже сегодня через чат-ботов можно заказать пиццу, посмотреть погоду, узнать свежие новости, забронировать столик в ресторане, назначить время на сеанс в парикмахерской и т.д.

Наша задача - разработать аналитический сервис статей из новостной ленты medium.com раздела **blockchain**, посвященного технологии блокчейн. В качестве пользовательского интерфейса будет служить мессенджер **Telegram**, который предоставляет интерфейс для создания и управления чат-ботами.

Функционал будет заключаться в следующем:

- Бот выводит основные тенденции в технологии blockchain, за указанный пользователем период.
- Пользователь вводит слово, бот выдает наиболее подходящую статью, связанную с введенным словом.

1 Методы и инструменты разработки

Необходимые инструменты для разработки аналитического сервиса с интерфейсом чат-бота:

1.1 Python

Python - высокоуровневый язык программирования общего назначения. В данный момент является одним из самых популярных языков программирования для анализа данных. Вся дальнейшая разработка будет вестись на языке Python.

1.2 SQLite3

SQLite3 - встраиваемая система управления базами данных. Необходима для хранения данных.

1.3 Natural Language Toolkit

NLTK - одна из наиболее популярных библиотек для обработки естественного языка.

Мы будем использовать две основные функции из этой библиотеки [1]: токенизация и лемматизация.

1.3.1 Токенизация

```
from nltk import word_tokenize
```

Токенизация - разбиение текста на осмысленные элементы

1.3.2 Лемматизация

```
from nltk import WordNetLemmatizer
```

Лемматизация - приведение слова к словарной форме

1.4 TF-IDF

Основным признаком для анализа текстов будет индекс TF-IDF.

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью

коллекции документов или корпуса. Вес некоторого слова пропорционален количеству употребления этого слова в документе, и обратно пропорционален частоте употребления слова в других документах коллекции [3].

Формула:

$$tf - idf(t, d, D) = \frac{n_t}{\sum_k n_k} \times \log \frac{|D|}{|\{d_i \in D | t \in d_i\}|}$$

где:

n_t - число вхождений слова t в документ

$\sum_k n_k$ - общее число слов в документе

|D| - число документов в коллекции

$\{d_i \in D | t \in d_i\}$ - число документов из коллекции D, в которых встречается слово t

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

1.5 Telegram API

Для начала следует разобраться что такое API

1.5.1 API

API (англ. application programming interface) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах [4].

Благодаря Telegram Bot API бот контроллируется https запросами.

1.5.2 pyTelegramBotAPI

pyTelegramBotAPI - библиотека [5], написанная на Python, для автоматизированной работы с Telegram Bot API

1.6 JSON

JSON - (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.
Обмен данными с Telegram Bot API происходит в формате JSON [6]. Хранить подготовленные данные в базе будем также в этом формате.
Синтаксис:
JSON-текст представляет собой (в закодированном виде) одну из двух структур:

- Набор пар *ключ: значение*. В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка, значением — любая форма.
- Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

В качестве значений в JSON могут быть использованы:

- Объект — это неупорядоченное множество пар *ключ:значение*, заключённое в фигурные скобки « ». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
- Массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми.
- Число.
- Литераллы true, false или null.
- Строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки.

Пример:

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
```

```
    "phoneNumbers": [
        "812 123-1234",
        "916 123-4567"
    ]
}
```

2 Этапы разработки

Перед написанием кода нужно убедиться, что следующие библиотеки установлены (версии указаны на момент написания работы):

- ***requests*** - 2.18.4 (для http/https запросов);
- ***PySocks*** - 1.6.8 (для использования протокола *SOCKS5*);
- ***bs4 (BeautifulSoup)*** - 0.0.1 (для парсинга веб-страниц);
- ***nltk*** - 3.2.5 (для обработки естественного языка);
- ***pyTelegramBotAPI*** - 3.6.3 (для автоматизированной работы с Telegram Bot API)

Также необходим аккаунт в мессенджере Telegram.

2.1 Создание бота

Чтобы зарегистрировать чат-бота нужно написать боту **@BotFather** комманду /**newbot** и следовать его инструкциям (установить имя, аватар, описание, стандартные комманды). Затем **BotFather** выдаст токен бота. С помощью этого токена полностью контроллируется бот. https://api.telegram.org/bot<token>/method - пример запроса к Telegram API. Допускаются **GET** и **POST** запросы. Со всеми методами можно ознакомиться на странице API https://core.telegram
methods
В частности нас интересуют методы **getUpdates** для получение обновлений, и **sendMessage** для отправки сообщений

2.2 Написание программы для парсинга новостной ленты

Для того, чтобы работать с данными, их нужно выкачать в базу. Т.к. medium.com не предоставляет API, что могло бы облегчить задачу, будем грубым методом их парсить с помощью класса ***BeautifulSoup*** библиотеки ***bs4.*** [7]

```
import sqlite3
import requests, requests.exceptions
from bs4 import BeautifulSoup

def parse_page(url):
    p_soup = BeautifulSoup(get_html(url), "html5lib")

    if check_lang(p_soup):

        try:
            title = p_soup.find('div', class_='section-inner sectionLayout--insetColumn'
        except AttributeError:
            title = ''

        date = p_soup.find('div', class_='ui-caption postMetaInline js-testPostMetaInlin
        .find('time').get('datetime')
        post_id = p_soup.find('div', class_='postArticle-content js-postField js-notesSo
        js-trackedPost').get('data-post-id')
        user_name = p_soup.find('a', class_='ds-link ds-link--styleSubtle ui-captionStro
        u-inlineBlock link link--darken link--darker').text
        user_id = p_soup.find('a', class_='ds-link ds-link--styleSubtle ui-captionStrong
                                          'u-inlineBlock link link--darken link--darker'
                                          .get('data-user-id')
        user_login = get_user_login(p_soup)
        text = parse_text(p_soup)

        article = {
            'title': title,
            'date': date,
            'post_id': post_id,
            'username': user_name,
```

```
            'user_id': user_id,
            'user_login': user_login,
            'text': text,
            'url': url
        }
        save_to_base(article)
```

```
def parse_text(soup):
    content = soup.find('div', class_='section-inner sectionLayout--insetColumn')
    tags = ['p', 'a', 'h2', 'h3']
    text = ''
    for tag in content.contents:
        if tag.name in tags:
            text += tag.text + ' '
        if tag.name == 'ul' or tag.name == 'ol':
            for li in tag.contents:
                text += li.text + ' '
    return text

def save_to_base(article):
    conn = sqlite3.connect('db.db3')
    cursor = conn.cursor()

    data = (article['title'], article['date'], article['post_id'], article['username'],
            article['user_id'], article['user_login'], article['text'], article['url'])

    if check_post_id(cursor, article['post_id']):
        cursor.execute("INSERT INTO articles (title, date, post_id, username, user_id,
        user_login, text, url) "
                       "VALUES (?, ?, ?, ?, ?, ?, ?, ?)", data)
        print('Succesfully parsed')
        conn.commit()

    conn.close()
```

В данном коде представлены три основные функции: парсинг отдельной веб-страницы, преобразование текстового содержимого и сохранение статьи в базу.

2.3 Подготовка данных к анализу

У нас уже есть данные в базе, но эти данные "сырые". Для того чтобы их можно было анализировать, нужно данные подготовить: разбить на токены и лемматизировать.

```
def lemmatize(quote):
    quote = word_tokenize(quote)
    quote = [w.lower() for w in quote]
    wnl = WordNetLemmatizer()
    quote = [wnl.lemmatize(w) for w in quote]

    i = 0
    while i < len(quote):
        if not quote[i].isalpha():
            quote.remove(quote[i])
        else:
            i += 1

    return json.dumps(quote)

def main(limit):
    conn = sqlite3.connect('db.db3')
    cursor = conn.cursor()
    cursor.execute("""SELECT * FROM articles ORDER BY id_articles
    DESC LIMIT ?""", (limit,))

    for row in cursor.fetchall():
        text = lemmatize(row[7])

        article = {
            'text': text,
            'date': row[2][:-5],
            'post_id': row[3]
        }
        save_to_base(cursor, article)

        conn.commit()

    conn.close()
```

2.4 Анализ данных

Сейчас в базе хранятся нормализованные данные в формате *JSON*. Теперь уже можем их анализировать.

```
def compute_tfidf(corpus):
    """Corpus - matrix"""
    def compute_tf(text):
        """TF"""
        """Returns Counter object"""
        tf_text = Counter(text)
        for i in tf_text:
            tf_text[i] = tf_text[i]/len(text)
        return tf_text

    def compute_idf(word, corp):
        """IDF"""
        return math.log10(len(corp)/sum(1 for i in corp if word in i))

    articles_list = []

    for article in corpus:
        tf_idf_dict = {}
        computed_tf = compute_tf(article)
        for word in computed_tf:
            tf_idf_dict[word] = computed_tf[word]*compute_idf(word, corpus)
        articles_list.append(OrderedDict(sorted(tf_idf_dict.items(), key=lambda t: t[1],

    return articles_list


def main(DATETIME):
    conn = sqlite3.connect('db.db3')
    cursor = conn.cursor()
    cursor.execute("""SELECT * FROM t_articles WHERE date(date) > date(?) LIMIT 50""", (

    corpus = []
```

```
    arr_article = []

    for row in cursor.fetchall():
        wrds = json.loads(row[1])
        if len(wrds) != 0:
            article = {
                'text': wrds,
                'date': row[2],
                'post_id': row[3]
            }
            arr_article.append(article)
            corpus.append(wrds)

    res = compute_tfidf(corpus)

    for article, b in zip(arr_article, res):
        article['text'] = json.dumps(b)

        save_to_base(cursor, article)
        conn.commit()

    conn.close()
```

Код выше сохраняет в базу данные в формате *ключ:значение*, где ключ - слово, а значение - его индекс *tf-idf* за определенный период. Данные упорядоченные в порядке убывания значения индекса.

2.5 Написание программы, управляющей ботом

```
bot = telebot.TeleBot(Token)
telebot.apihelper.proxy = {'https': Proxy}
wnl = WordNetLemmatizer()

@bot.message_handler(content_types=['text'])
def handle_text(message):
    if message.text == '1 DAY':
```

```
        answer = content.get_words(datetime.now() - timedelta(days=1))
        bot.send_message(message.from_user.id, answer)
        log(message, answer)
    elif message.text == '2 DAYS':
        answer = content.get_words(datetime.now() - timedelta(days=2))
        bot.send_message(message.from_user.id, answer)
        log(message, answer)
    elif message.text == '3 DAYS':
        answer = content.get_words(datetime.now() - timedelta(days=3))
        bot.send_message(message.from_user.id, answer)
        log(message, answer)
    else:
        answer = content.get_article(datetime.now() - timedelta(days=1), wnl.lemmatize(m
        bot.send_message(message.from_user.id, answer)
        log(message, answer)


bot.polling(none_stop=True, interval=0)
```

Благодаря параметру *none_stop=True* метода *polling* бот в каждый момент времени получает обновления (информацию о запросах), и обрабатывает их, вызывая соответственные функции генерации ответа.

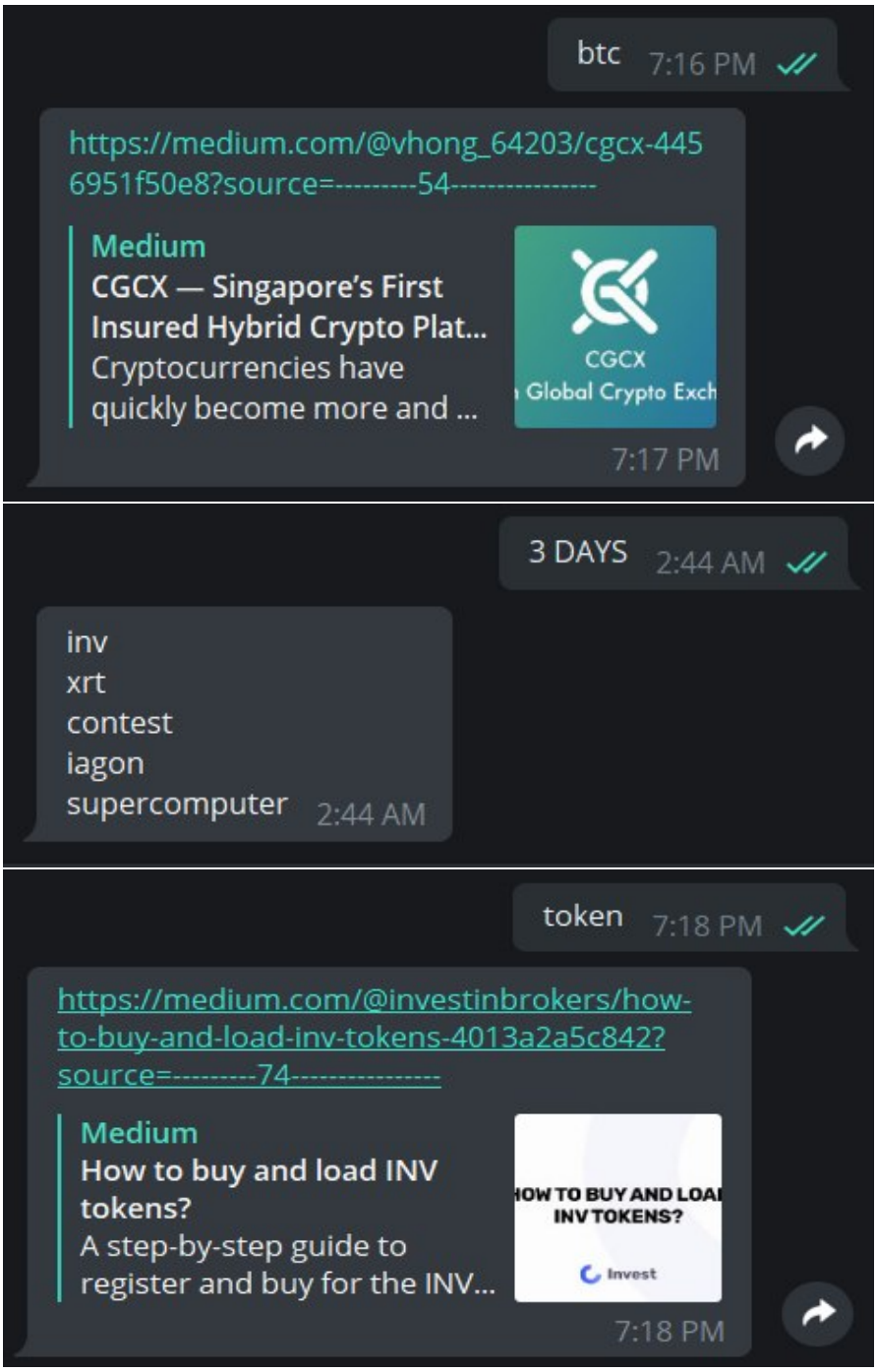
```
def get_words(DATETIME):
    conn = sqlite3.connect('db.db3')
    cursor = conn.cursor()
    cursor.execute("""SELECT * FROM tf_articles WHERE date(date) > date(?) LIMIT 50""",

    words = {}

    for row in cursor.fetchall():
        counter = 0
        for k, v in json.loads(row[1]).items():
            if counter < 2:
                counter += 1
                if len(words) > 4:
```

```
                    for i in list(words):
                        if v > words[i]:
                            words[k] = v
                            del words[i]

                else:
                    words[k] = v
    return '\n'.join(words)


def get_article(DATETIME, word):
    conn = sqlite3.connect('db.db3')
    cursor = conn.cursor()
    cursor.execute("""SELECT * FROM tf_articles WHERE date(date) > date(?)
    LIMIT 50""", (DATETIME,))
    value = 0
    post_id = 0
    for row in cursor.fetchall():
        tfidf_dict = json.loads(row[1])
        if word in tfidf_dict:
            if tfidf_dict[word] > value:
                value = tfidf_dict[word]
                post_id = row[3]

    if value != 0:
        cursor.execute("""SELECT * FROM articles WHERE post_id=?""", (post_id, ))
        link = cursor.fetchone()[8]
        return link
    return 'No such articles'
```

Выше представлены функции, генерирующие ответ:

- ***get_words*** принимает параметром промежуток времени, и возвращает несколько слов с наибольшем индексом *tf-idf* за указанный период;
- ***get_articles*** принимает параметром промежуток времени и слово, возвращает ссылку на наиболее подходящую статью, связанную с введенным словом.

3 Результат



4 Заключение

Целью разработки была аналитическая система в удобном интерфейсе мессенджера Telegram. Во время разработки мы столкнулись с такими проблемами: новостной портал medium.com не предоставляет API, следовательно подготовка данных к анализу потребовала больше усилий,

чем ожидалось; т. к. в указанном портале пользователи имеют возможность писать статьи на разных языках, индекс TF-IDF для таких слов не является объективным.
В будущем можно расширить функционал системы, увеличив число источников информации и расширив тематику для агрегации новостей. Также имеет смысл рассматривать не только отдельные слова, но и словосочетания.

Приложение

github.com/JJBT/Sci-res - исходный код на GitHub.

Список литературы

[1] *Bird, Steven, Edward Loper and Ewan Klein* (2009). Natural Language Processing with Python. O'Reilly Media Inc.

[2] Блокчейн // Википедия. [2015—2018]. Дата обновления: 28.04.2018. URL: https://ru.wikipedia.org/?oldid=92356552 (дата обращения: 28.04.2018).

[3] TF-IDF // Википедия. [2006—2018]. Дата обновления: 14.05.2018. URL: https://ru.wikipedia.org/?oldid=92651780 (дата обращения: 14.05.2018).

[4] API // Википедия. [2005—2018]. Дата обновления: 23.05.2018. URL: https://ru.wikipedia.org/?oldid=92827248 (дата обращения: 23.05.2018).

[5] pyTelegramBotAPI Documentation // GitHub. [2015-2018]. Дата обновления: 26.05.2018. URL: https://github.com/eternnoir/pyTelegramBotAPI (дата обращения: 10.05.2018)

[6] JSON // Википедия. [2006—2018]. Дата обновления: 22.05.2018. URL: https://ru.wikipedia.org/?oldid=92818562 (дата обращения: 22.05.2018).

[7] Beautiful Soup Documentation // Crummy. [2004-2015]. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата обращения: 20.03.2018)