自然言語処理

期末課題：chatbot

提出日：　　平成 31 年 2 月 11 日（月）
提出期限：　平成 31 年 2 月 11 日（月）
学籍番号：　2516290929
氏名：　　森田聡太

1. 目的

自然言語処理の講義内で学習した知識や技術を総合的に用いたプログラムを作成することで知識・技術の関連性を含めたプログラムを作成できるようになる.

2. 課題内容

Python 言語を使ってプログラミングを行い, Mastodon 上で会話を行う bot を作成せよ.

- 自分のアカウント宛ての tweet に対して，retweet を行うこと.
- この講義で習得した知識や技術を用いること.

  例：
  - 形態素解析を用いた tweet 理解
  - 言語モデル（N-gram，オートマトン）を用いた文生成
  - 情報検索を行って質問に回答
- 10 秒間に 1 回しか tweet を返さないようにするなど，何らかの遅延を組み込むこと.

3. 概要

bot 宛にツイートされた内容の形態素解析を行い，その内容に準じた最新のニュースの要約と見出し，URL を retweet で返信する.

4. 処理手順

以下に大まかな処理手順を示す.

i. 自分宛の tweet を判断し，tweet 内容を読みこむ.
ii. 読み込んだ内容の形態素解析を行い情報（ニュース）検索に必要と思われる名詞を取り出す.
iii. 取り出された名詞を用いてニュース検索を行う.
※検索手法としてはスクレイピングを用いる.
iv. 検索されたニュース記事を要約する.
v. 検索結果上位 1 件の見出し名，URL を retweet として返信する.

5. ソースコード

使用したソースコードをソースコード 1 ～ 5 に示す．なお，ソースコードの役割としては,

- mstdn_bot.py
  bot の動作をつかさどっているメインのプログラム.
- libmstdn.py
  Mastodon での内部の情報の取得，送信関連のプログラム.

- get_news.py
  形態素解析後の自分宛の tweet の内容に応じてニュース記事を取得するプログラム.
- morphological_analysis.py
  形態素解析を行う.
- get_summary.py
  文字列を投げるとその要約を返すプログラム.

ソースコード 1 ：mstdn_bot.py

```
import sys
import re
import libmstdn
import get_news
import morphological_analysis
import get_summary
import time


#Mastodon ホスト
#注意："https://"を記述せず,ホスト名のみ記述すること
MASTODON_HOST = "memphis.ibe.kagoshima-u.ac.jp"

#MastodonAPI アクセストークン
#Mastodon のサイトでアプリの登録を行い，取得したアクセストークンを記述すること.
ACCESS_TOKEN = "2847a7d24f17b304fc252506077aad2f69454f15d1ccdea541334f39497e7535"

def remove_html_tags(content):
    """
    文字列内の HTML タグを削除

    Args:
        content(str):HTML を削除する文字列

    Returns:
        str:HTML を削除された文字列
    """
    return re.sub("<[^>]*>", "", content).strip()

def is_to_me(status, my_id):
    """
    自分へのリプライかどうかを判定

    Args:
        status(dict):トゥート情報を格納した dict
        my_id(int):自分のアカウント ID

    Returns:
        bool:True であれば自分へのリプライ
    """

    for mention in status["mentions"]:
        if mention["id"] == my_id:
```

```
44          return True
45      return False
46
47  def generate_reply(status, my_name):
48      """
49      リプライを生成
50
51      Args:
52          status(dict):
53          トゥート情報を格納した dict
54
55      Returns:
56          str:生成されたリプライ
57      """
58      #tweet 内容の取り出し
59      received_text = remove_html_tags(status["content"])
60      received_text = "".join(received_text.split()[1:])
61      #tweet 内容の形態素解析
62      mor_ana = morphological_analysis.GetNoun(received_text)
63      #print(mor_ana.get_words())
64      received_text = "".join(mor_ana.get_words())
65      if received_text == "":
66          return "対象のニュースはありません．"
67      toot_form = status["account"]["username"]
68
69      #retweet するニュースを取得
70      toot = get_news.GetNews(received_text)
71      toot.search_news()
72      #ニュースの要約
73      summ = get_summary.GetSummary(toot.readable_article)
74      summ.get_summary()
75      return "[要約]" + str(summ.summary) + "¥n¥n" + "【" + toot.news_title[0] + "】" + toot.news_url[0]
76
77
78      """
79      if "こんにちは" in received_text:
80          return toot_form + "さん，こんにちは！"
81      elif "こんばんは" in received_text:
82          return toot_form + "さん，こんにちは！"
83      elif "はじめまして" in received_text:
84          return toot_form + "さん，はじめまして！私は" + my_name + "です！"
85      else:
86          return "ごめんなさい,よくわかりません．"
87      """
88  def main():
89      api = libmstdn.MastodonAPI(
90              mastodon_host=MASTODON_HOST,
91              access_token=ACCESS_TOKEN)
92
93      account_info = api.verify_account()
94      my_id = account_info["id"]
95      my_name = account_info["username"]
96      print("Started bot, name: {}, id: {}".format(my_name, my_id))
97
98      stream = api.get_user_stream()
99      for status in stream:
100         if is_to_me(status, my_id):
```

```
101         received_text = remove_html_tags(status["content"])
102         toot_id = status["id"]
103         toot_from = status["account"]["username"]
104         print("received from {}: {}".format(toot_from, received_text))
105 
106         reply_text = "@{} {}".format(
107             toot_from, generate_reply(status, my_name))
108         api.toot(reply_text, toot_id)
109         print("post to {}: {}".format(toot_from, reply_text))
110         time.sleep(10)
111     return 0
112 
113 if __name__ == "__main__":
114     sys.exit(main())
```

ソースコード 2 ：libmstdn.py

```
 1 import json
 2 import requests
 3 
 4 
 5 class MastodonAPIError(Exception):
 6     """ MastodonAPI のエラー """
 7 
 8     def __init__(self, message, http_status=None):
 9         """
10         初期化
11 
12         Args:
13             message (str): 例外メッセージ
14             http_status (int): HTTP ステータスコード
15         """
16 
17         if http_status is not None:
18             message = "http_status: {}, response: {}".format(
19                 http_status, message)
20         else:
21             message = "response: {}".format(message)
22 
23         super(MastodonAPIError, self).__init__(message)
24 
25 
26 class MastodonStream:
27     """ Mastofon のストリームを受信するクラス """
28 
29     def __init__(self, url, access_token=None):
30         """
31         初期化
32 
33         Args:
34             url (str): ストリーム API の URL
35             access_token (str): API のアクセストークン
36         """
37         self.__url = url
38         self.__access_token = access_token
```

```
39
40      def __connect(self):
41          """ ストリームに接続 """
42          headers = {}
43          if self.__access_token is not None:
44              headers["Authorization"] = "Bearer " + self.__access_token
45
46          self.__stream = requests.get(self.__url, headers=headers, stream=True)
47
48          if self.__stream.status_code != 200:
49              raise MastodonAPIError(
50                  self.__stream.text, self.__stream.status_code)
51
52          self.__stream_gen = self.__stream.iter_lines()
53          self.__event_type = ""
54
55      def __iter__(self):
56          """ ストリームデータを取得するイテレータを取得する """
57          self.__connect()
58          return self
59
60      def __next__(self):
61          """
62          次のストリームデータを取得する
63
64          この関数は，次のデータを受信するまで待機する．
65          また，データは１行ごとに返される．
66
67          Returns:
68              str: 取得されたデータ
69          """
70
71          while True:
72              try:
73                  line = next(self.__stream_gen).decode("utf-8").strip()
74              except StopIteration:
75                  # 接続が切れたときは再接続する
76                  self.__connect()
77                  continue
78
79              line_kv = [d.strip() for d in line.split(":", 1)]
80              if len(line_kv) < 2:
81                  continue
82              line_key = line_kv[0]
83              line_value = line_kv[1]
84
85              if line_key == "event":
86                  self.__event_type = line_value
87              elif line_key == "data":
88                  if self.__event_type == "update":
89                      return json.loads(line_value)
90                  elif self.__event_type == "notification":
91                      return json.loads(line_value)["status"]
92              else:
93                  pass
94
95
```

```python
class MastodonAPI:
    """ Mastodon の API を叩くクラス """

    def __init__(self, mastodon_host, access_token):
        """
        初期化

        Args:
            mastodon_host (str): Mastodon インスタンスのホスト名
            access_token (str): API のアクセストークン
        """
        self.__host = mastodon_host
        self.__access_token = access_token

    def verify_account(self):
        """
        アカウント情報を取得

        Returns:
            dict: アカウント情報を格納した dict
        """
        url = "https://{}/api/v1/accounts/verify_credentials".format(self.__host)
        resp = requests.get(url, headers=self.__auth_header(), timeout=10)
        if resp.status_code != 200:
            raise MastodonAPIError(resp.text, resp.status_code)
        return resp.json()

    def toot(self, status, in_reply_to_id=None):
        """
        トゥートを送信

        Args:
            status (str): トゥートの内容
            in_reply_to_id (int): リプライするトゥートの ID

        Returns:
            dict: 送信されたトゥート情報を格納した dict
        """
        url = "https://{}/api/v1/statuses".format(self.__host)

        data = {"status": status}
        if in_reply_to_id is not None:
            data["in_reply_to_id"] = in_reply_to_id

        resp = requests.post(
            url, headers=self.__auth_header(), data=data, timeout=10)
        if resp.status_code != 200:
            raise MastodonAPIError(resp.text, resp.status_code)
        return resp.json()

    def get_public_stream(self):
        """
        連合タイムラインのストリームを取得

        Returns:
            MastodonStream: ストリームクラス
        """
```

```
153             return MastodonStream(
154                 url="https://{}/api/v1/streaming/public".format(self.__host))
155 
156     def get_user_stream(self):
157         """
158         ユーザタイムラインのストリームを取得
159 
160         Returns:
161             MastodonStream: ストリームクラス
162         """
163         return MastodonStream(
164             url="https://{}/api/v1/streaming/user".format(self.__host),
165             access_token=self.__access_token)
166 
167     def __auth_header(self):
168         """OAuth2 ヘッダを生成 """
169         return {"Authorization": "Bearer " + self.__access_token}
```

ソースコード 3 ：get_news.py

```
 1 import feedparser
 2 import json
 3 import pprint
 4 import urllib.request
 5 from readability.readability import Document
 6 import re
 7 
 8 
 9 class GetNews:
10     #news_url #検索結果 url_クラス変数
11     def __init__(self, search_str):
12         # RSS のスクレイピング
13 
14         self._search_str = search_str
15 
16     def format_text(text):
17         text=re.sub(r'https?://[¥w/:%#¥$&¥?¥(¥)~¥.=¥+¥-…]+', "", text)
18         text=re.sub('RT', "", text)
19         text=re.sub('お気に入り', "", text)
20         text=re.sub('まとめ', "", text)
21         text=re.sub(r'[!-~]', "", text)#半角記号,数字,英字
22         text=re.sub(r'[：-@]', "", text)#全角記号
23         text=re.sub('¥n', " ", text)#改行文字
24         return text
25 
26 
27     def search_news(self):
28         # URL エンコーディング
29         s_quote = urllib.parse.quote(self._search_str)
30         #Google ニュース
31         url = "https://news.google.com/news/rss/search/section/q/" + s_quote + "/" + s_quote +
"?ned=jp&amp;hl=ja&amp;gl=JP"
32         #url = "http://www.google.co.jp/search?hl=jp&gl=JP&num=10&q=" + s_quote
33 
34         d = feedparser.parse(url)
```

```
35            news = list()
36
37            for i, entry in enumerate(d.entries, 1):
38
39                p = entry.published_parsed
40                sortkey = "%04d%02d%02d%02d%02d%02d" % (p.tm_year, p.tm_mon, p.tm_mday, p.tm_hour,
p.tm_min, p.tm_sec)
41
42                tmp = {
43                    "no": i,
44                    "title": entry.title,
45                    "link": entry.link,
46                    "published": entry.published,
47                    "sortkey": sortkey
48                }
49
50                news.append(tmp)
51
52            self.news_title = [d.get('title') for d in news]
53            self.news_url = [d.get('link') for d in news]
54
55            #HTML テキストを取得
56            url_ = self.news_url[0]
57
58            html = urllib.request.urlopen(url_).read()
59            _readable_article = Document(html).summary()
60            self.readable_title = Document(html).short_title()
61            cleanr = re.compile('<.*?>')
62            self.readable_article = re.sub(cleanr, '', _readable_article)
63            print(self.readable_article)
64
```

ソースコード 4 ：morphological_analysis.py

```
1 import MeCab
2
3 class GetNoun:
4
5   def __init__(self, text):
6     self.mecab = MeCab.Tagger("-Ochasen")
7     self.mecab.parse("")
8     self.sentence = text
9
10  # MeCab を使って形態素解析
11  def ma_parse(self, sentence):
12    node = self.mecab.parseToNode(sentence)
13    while node:
14      if node.feature.startswith("名詞"):
15        yield node.surface
16      node = node.next
17
18  #文字列取得
19  def get_words(self):
20    words =  [word for word in self.ma_parse(self.sentence)]
21    return words
```

ソースコード５：get_summary.py

```
from janome.analyzer import Analyzer
from janome.charfilter import UnicodeNormalizeCharFilter, RegexReplaceCharFilter
from janome.tokenizer import Tokenizer as JanomeTokenizer  # sumy の Tokenizer と名前が被るため
from janome.tokenfilter import POSKeepFilter, ExtractAttributeFilter
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lex_rank import LexRankSummarizer

class GetSummary:
    def __init__(self, text):
        self._text = text

    def get_summary(self):
        # 1 行 1 文となっているため、改行コードで分離
        sentences = [t for t in self._text.split('¥n')]
        for i in range(1):
            print(sentences[i])

        # 形態素解析器を作る
        analyzer = Analyzer(
            [UnicodeNormalizeCharFilter(), RegexReplaceCharFilter(r'[(¥)「」、。]', ' ')],  # () 「」、。は全てスペースに置き換える
            JanomeTokenizer(),
            [POSKeepFilter(['名詞', '形容詞', '副詞', '動詞']), ExtractAttributeFilter('base_form')]  # 名詞・形容詞・副詞・動詞の原型のみ
        )

        # 抽出された単語をスペースで連結
        # 末尾の'。'は、この後使う tinysegmenter で文として分離させるため。
        corpus = [' '.join(analyzer.analyze(s)) + '。' for s in sentences]
        """
        for i in range(2):
            print(corpus[i])
        """

        # 連結した corpus を再度 tinysegmenter でトークナイズさせる
        parser = PlaintextParser.from_string(''.join(corpus), Tokenizer('japanese'))

        # LexRank で要約を 2 文抽出
        summarizer = LexRankSummarizer()
        summarizer.stop_words = [' ']  # スペースも 1 単語として認識されるため、ストップワードにすることで除外する

        self.summary = summarizer(document=parser.document, sentences_count=2)

        # 元の文を表示
        for sentence in self.summary:
            print(sentences[corpus.index(sentence.__str__())])
```

6. プログラム概要

i. mstdn_bot.py

A) is_to_me(status, my_id)

自分へのリプライかどうかを判定する．判定方法としてはメンションされている id と自身の id を見比べている．

B) generate_reply(status, my_name)

リプライの内容を生成している．

59-60 行目：tweet 内容の取り出し．

62-64 行目：取り出した tweet 内容から形態素解析を用いて名詞を取り出す．

70-71 行目：ニュースを検索（スクレイピング）して取得．

73-74 行目：ニュースの要約を取得

C) main()

libmstdn を利用して API 関連を処理しリプライの取得 tweet を行っている．

ii. libmstdn.py

class MastodonAPIError(Exception)

MastodonAPI のエラーを処理するクラス

A) __init__(self, message, http_status=None)

初期化関数．例外の処理を行っている．

Class MastodonStream

Mastofon のストリームを受信するクラス

A) __init__(self, url, access_token=None)

初期化関数．ストリーム API の URL と API のアクセストークンをインスタンス変数として初期化．

B) __connect(self)

ストリームに接続する関数

C) __iter__(self)

ストリームデータを取得するイテレータを取得する

D) __next(self)__

次のストリームデータを取得する関数．データの取得をここで判定している．

Class MastodonAPI

Mastodon の API を叩くクラス

A) __init__(self, mastodon_host, access_token)

初期化関数．Mastodon インスタンスのホスト名と API のアクセストークンを初期化．

B) verify_account(self)

アカウント情報を取得する関数．

C） toot(self, status, in_reply_to_id=None)
実際にトゥートの制御を行う関数．トゥートの内容とリプライするトゥートの id を受け取りトゥートをする．

D） get_public_stream(self)
連合タイムラインのストリームを取得する関数

E） get_user_stream(self)
ユーザタイムラインのストリームを取得する関数

F） __auth_header(self)
OAuth2 ヘッダを生成する関数

iii. get_news.py

class GetNews
ニュース記事を検索（スクレイピング）するクラス

A） __init__(self, search_str)
初期化関数．スクレイピング（検索）する文字列を受け取り初期化する．

B） format_text(text)
取得したサイトの文字列から HTML タグを削除したりなど整列を行う関数．

C） search_news(self)
url で検索対象を設定し，検索結果を複数得る．news_title に取得したニュースのタイトル，news_url に取得したニュースの URL を格納する．

iv. morphological_analysis.py

class GetNoun
リプライされた文字列の形態素解析を行うクラス

A） __init__(self, text)
初期化関数．形態素解析を行う文字列を初期化．

B） ma_parse(self, sentence)
形態素解析を行っている関数．

C） get_words(self)
形態素解析を行った結果を得ている関数．

v. get_summary.py

class GetSummary
ニュース記事の要約を行うクラス

A） __init__(self, text)
初期化関数．要約を行う文字列を初期化．

B） get_summary(self)
形態素解析は janome を用いて行い，要約は LexRank を用いる．

7. 実行例

実行例として図１～４を示す.

図１．実行例１

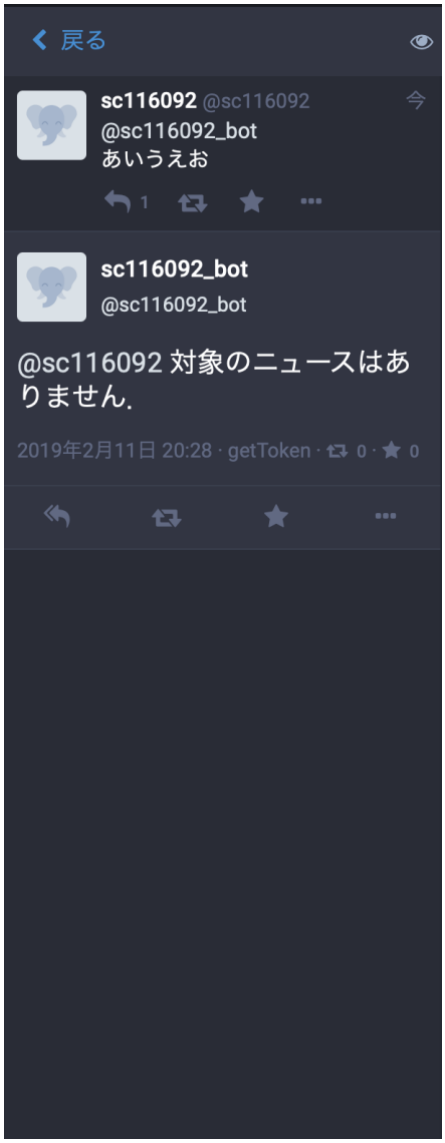

図２．実行例２

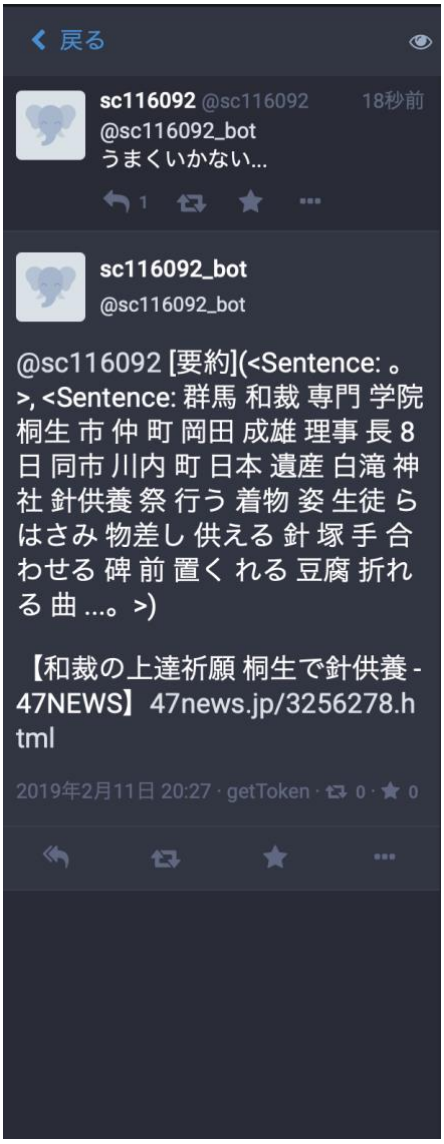

図３．実行例３

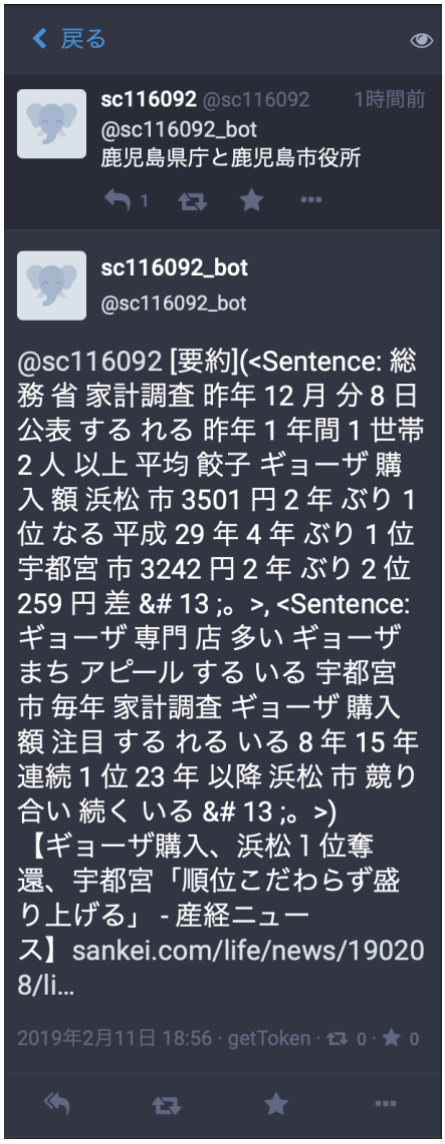

図４．実行例４

8. 考察

実行例３，４では一応ニュー検索と要約ができているが，要約内容で”(<Sentence:”など普通の言葉使いでは出てこないような文字列が含まれてしまっている．これは get_news.py の GetNews クラスの format_text 関数で文字列の整列がうまく行っていないからである．整列方法として文字列を正規表現で整理しているがこの方法がうまく行っていない．また，トゥート内容とリプライとの一貫性があまり内容にも思われる．これはトゥートの文字数制限があるため詳しい結果は求められないと思われる．

図１の実行例からは要約がうまくいっていない．これは取り出したサイト内の本文量が少なかったことが要因と思われる．図２では名詞が取り出せなかったことによりニュース結果が表示できていない．これらのようにトゥート内容とニュースに大きく依存してしま

う．加えて実行例として示せていないが，ニュース検索結果が取得できない可能性も見られる．この依存性にも対応できるようにトゥート内容から予測してニュースが存在するかの判定を組み込めたらより良いシステムになったと考える．

9. 感想

自然言語処理の期末課題として取り組んだが，自然言語処理という分野は理解がとても難しかったと改めて思った．雰囲気としては理解することが簡単だが，実際にプログラムを組むとなると訳が違ってきた．今回の課題でもライブラリを多用しているのでライブラリに頼らずに構築できたら真に理解できたと言えると感じた．

10. 参考文献

[1] https://ymgsapo.com/scraping-google-news/（最終閲覧日：2019-02-11）
[2] http://opendata.jp.net/?p=6120（最終閲覧日：2019-02-11）
[3] https://ohke.hateblo.jp/entry/2018/11/17/230000（最終閲覧日：2019-02-11）
[4] https://remotestance.com/blog/129/（最終閲覧日：2019-02-11）