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

自然言語処理

期末課題：chatbot

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．

1. 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．

1. 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．

1. 処理手順

　以下に大まかな処理手順を示す．

1. 自分宛のtweetを判断し，tweet内容を読みこむ．
2. 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．
3. 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．

1. 検索されたニュース記事を要約する．
2. 検索結果上位1件の見出し名，URLをretweetとして返信する．
3. 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

1 **import** **sys**

2 **import** **re**

3 **import** **libmstdn**

4 **import** **get\_news**

5 **import** **morphological\_analysis**

6 **import** **get\_summary**

7 **import** **time**

8

9

10 *#Mastodonホスト*

11 *#注意："https://"を記述せず,ホスト名のみ記述すること*

12 MASTODON\_HOST = "memphis.ibe.kagoshima-u.ac.jp"

13

14 *#MastodonAPIアクセストークン*

15 *#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.*

16 ACCESS\_TOKEN = "2847a7d24f17b304fc252506077aad2f69454f15d1ccdea541334f39497e7535"

17

18 **def** remove\_html\_tags(content):

19 *"""*

20  *文字列内のHTMLタグを削除*

21

22  *Args:*

23  *content(str):HTMLを削除する文字列*

24

25  *Returns:*

26  *str:HTMLを削除された文字列*

27  *"""*

28 **return** re.sub("<[^>]\*>", "", content).strip()

29

30 **def** is\_to\_me(status, my\_id):

31 *"""*

32  *自分へのリプライかどうかを判定*

33

34  *Args:*

35  *status(dict):トゥート情報を格納したdict*

36  *my\_id(int):自分のアカウントID*

37

38  *Returns:*

39  *bool:Trueであれば自分へのリプライ*

40  *"""*

41

42 **for** mention **in** status["mentions"]:

43 **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

96 **class** **MastodonAPI**:

97 *""" MastodonのAPIを叩くクラス """*

98

99 **def** \_\_init\_\_(self, mastodon\_host, access\_token):

100 *"""*

101  *初期化*

102

103  *Args:*

104  *mastodon\_host (str): Mastodonインスタンスのホスト名*

105  *access\_token (str): APIのアクセストークン*

106  *"""*

107 self.\_\_host = mastodon\_host

108 self.\_\_access\_token = access\_token

109

110 **def** verify\_account(self):

111 *"""*

112  *アカウント情報を取得*

113

114  *Returns:*

115  *dict: アカウント情報を格納したdict*

116  *"""*

117 url = "https://{}/api/v1/accounts/verify\_credentials".format(self.\_\_host)

118 resp = requests.get(url, headers=self.\_\_auth\_header(), timeout=10)

119 **if** resp.status\_code != 200:

120 **raise** MastodonAPIError(resp.text, resp.status\_code)

121 **return** resp.json()

122

123 **def** toot(self, status, in\_reply\_to\_id=None):

124 *"""*

125  *トゥートを送信*

126

127  *Args:*

128  *status (str): トゥートの内容*

129  *in\_reply\_to\_id (int): リプライするトゥートのID*

130

131  *Returns:*

132  *dict: 送信されたトゥート情報を格納したdict*

133  *"""*

134 url = "https://{}/api/v1/statuses".format(self.\_\_host)

135

136 data = {"status": status}

137 **if** in\_reply\_to\_id **is** **not** None:

138 data["in\_reply\_to\_id"] = in\_reply\_to\_id

139

140 resp = requests.post(

141 url, headers=self.\_\_auth\_header(), data=data, timeout=10)

142 **if** resp.status\_code != 200:

143 **raise** MastodonAPIError(resp.text, resp.status\_code)

144 **return** resp.json()

145

146 **def** get\_public\_stream(self):

147 *"""*

148  *連合タイムラインのストリームを取得*

149

150  *Returns:*

151  *MastodonStream: ストリームクラス*

152  *"""*

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

1 **from** **janome.analyzer** **import** Analyzer

2 **from** **janome.charfilter** **import** UnicodeNormalizeCharFilter, RegexReplaceCharFilter

3 **from** **janome.tokenizer** **import** Tokenizer **as** JanomeTokenizer *# sumyのTokenizerと名前が被るため*

4 **from** **janome.tokenfilter** **import** POSKeepFilter, ExtractAttributeFilter

5 **from** **sumy.parsers.plaintext** **import** PlaintextParser

6 **from** **sumy.nlp.tokenizers** **import** Tokenizer

7 **from** **sumy.summarizers.lex\_rank** **import** LexRankSummarizer

8

9 **class** **GetSummary**:

10 **def** \_\_init\_\_(self, text):

11 self.\_text = text

12

13 **def** get\_summary(self):

14 *# 1行1文となっているため、改行コードで分離*

15 sentences = [t **for** t **in** self.\_text.split('**\n**')]

16 **for** i **in** range(1):

17 **print**(sentences[i])

18

19 *# 形態素解析器を作る*

20 analyzer = Analyzer(

21 [UnicodeNormalizeCharFilter(), RegexReplaceCharFilter(r'[(\)「」、。]', ' ')], *# ()「」、。は全てスペースに置き換える*

22 JanomeTokenizer(),

23 [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*

24 )

25

26 *# 抽出された単語をスペースで連結*

27 *# 末尾の'。'は、この後使うtinysegmenterで文として分離させるため。*

28 corpus = [' '.join(analyzer.analyze(s)) + '。' **for** s **in** sentences]

29 *"""*

30  *for i in range(2):*

31  *print(corpus[i])*

32  *"""*

33

34 *# 連結したcorpusを再度tinysegmenterでトークナイズさせる*

35 parser = PlaintextParser.from\_string(''.join(corpus), Tokenizer('japanese'))

36

37 *# LexRankで要約を2文抽出*

38 summarizer = LexRankSummarizer()

39 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*

40

41 self.summary = summarizer(document=parser.document, sentences\_count=2)

42

43 *# 元の文を表示*

44 **for** sentence **in** self.summary:

45 **print**(sentences[corpus.index(sentence.\_\_str\_\_())])

1. プログラム概要
2. mstdn\_bot.py
3. 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．

1. 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

1. 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．

1. libmstdn.py

class MastodonAPIError(Exception)

MastodonAPIのエラーを処理するクラス

1. \_\_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

1. \_\_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．

1. \_\_connect(self)

ストリームに接続する関数

1. \_\_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

1. \_\_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

1. \_\_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．

1. verify\_account(self)

アカウント情報を取得する関数．

1. 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．

1. 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

1. 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

1. \_\_auth\_header(self)

OAuth2ヘッダを生成する関数

1. 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

1. \_\_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．

1. 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．

1. 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．

1. 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

1. \_\_init\_\_(self, text)

初期化関数．形態素解析を行う文字列を初期化．

1. ma\_parse(self, sentence)

形態素解析を行っている関数．

1. get\_words(self)

形態素解析を行った結果を得ている関数．

1. get\_summary.py

class GetSummary

ニュース記事の要約を行うクラス

1. \_\_init\_\_(self, text)

初期化関数．要約を行う文字列を初期化．

1. get\_summary(self)

形態素解析はjanomeを用いて行い，要約はLexRankを用いる．

1. 実行例

　実行例として図１〜４を示す．

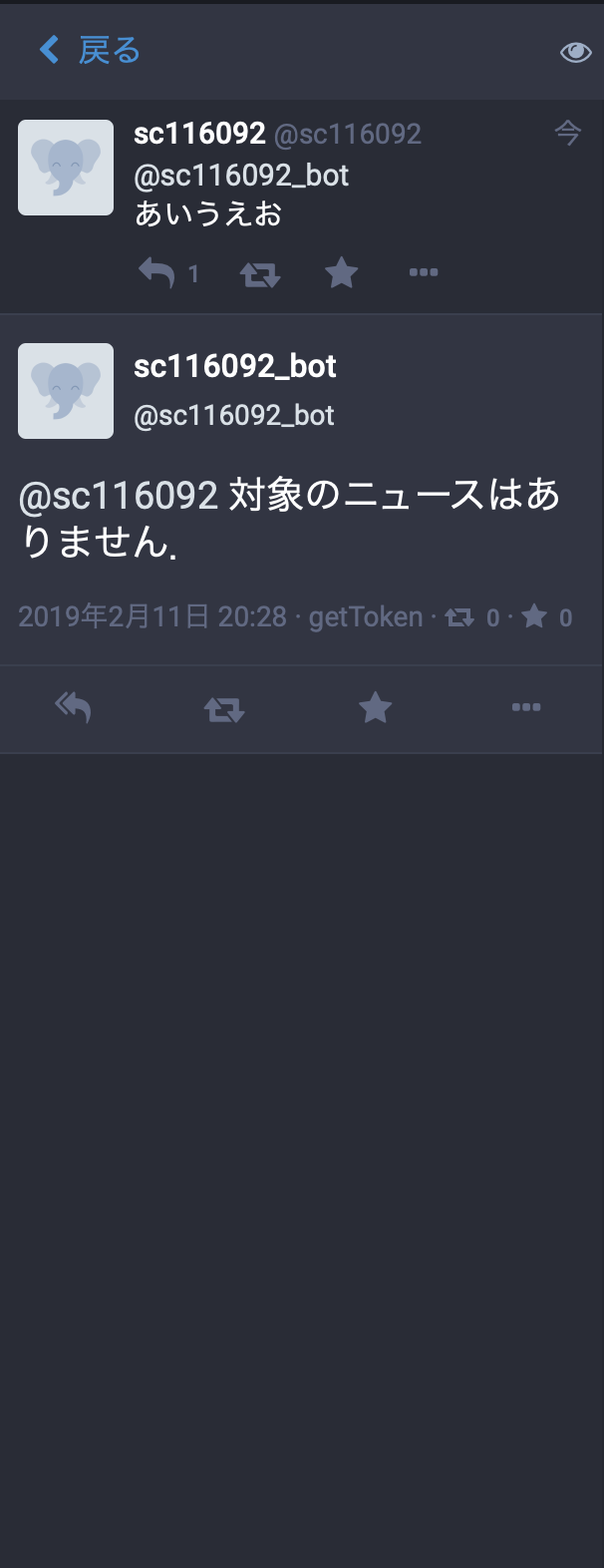
 

図１．実行例１　　　　　　　　　　　図２．実行例２

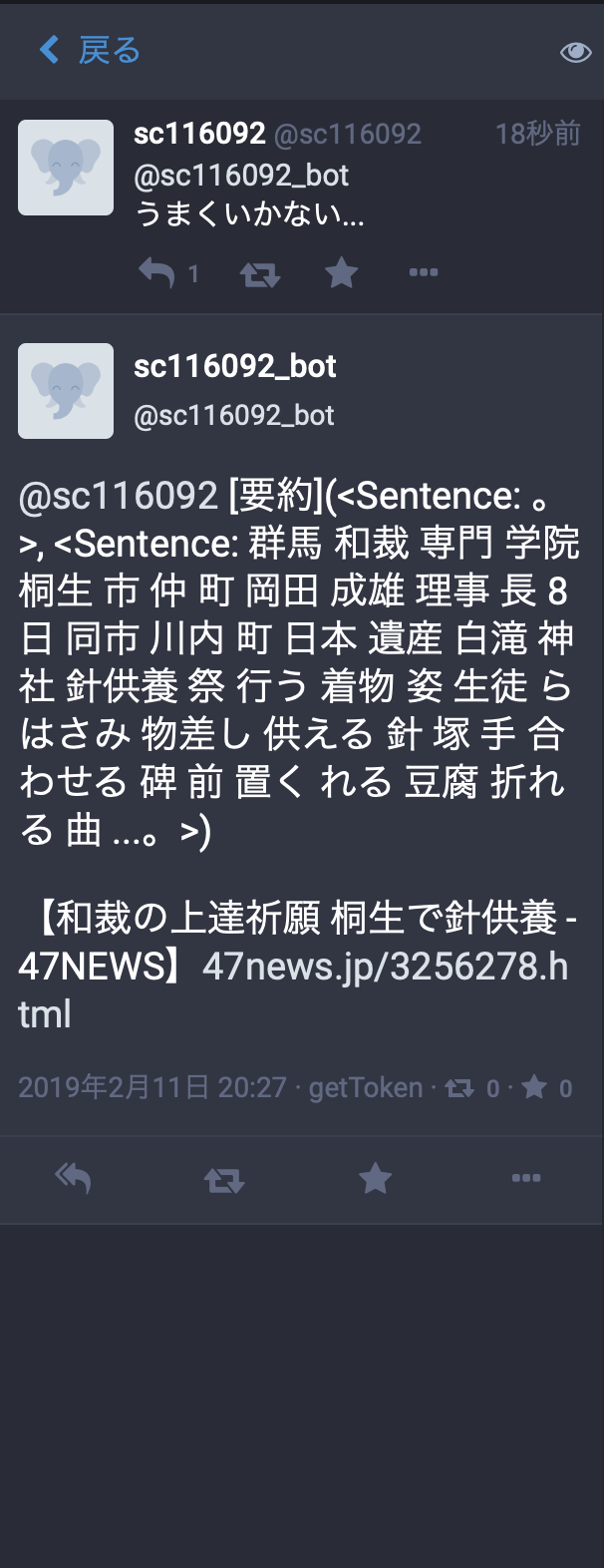
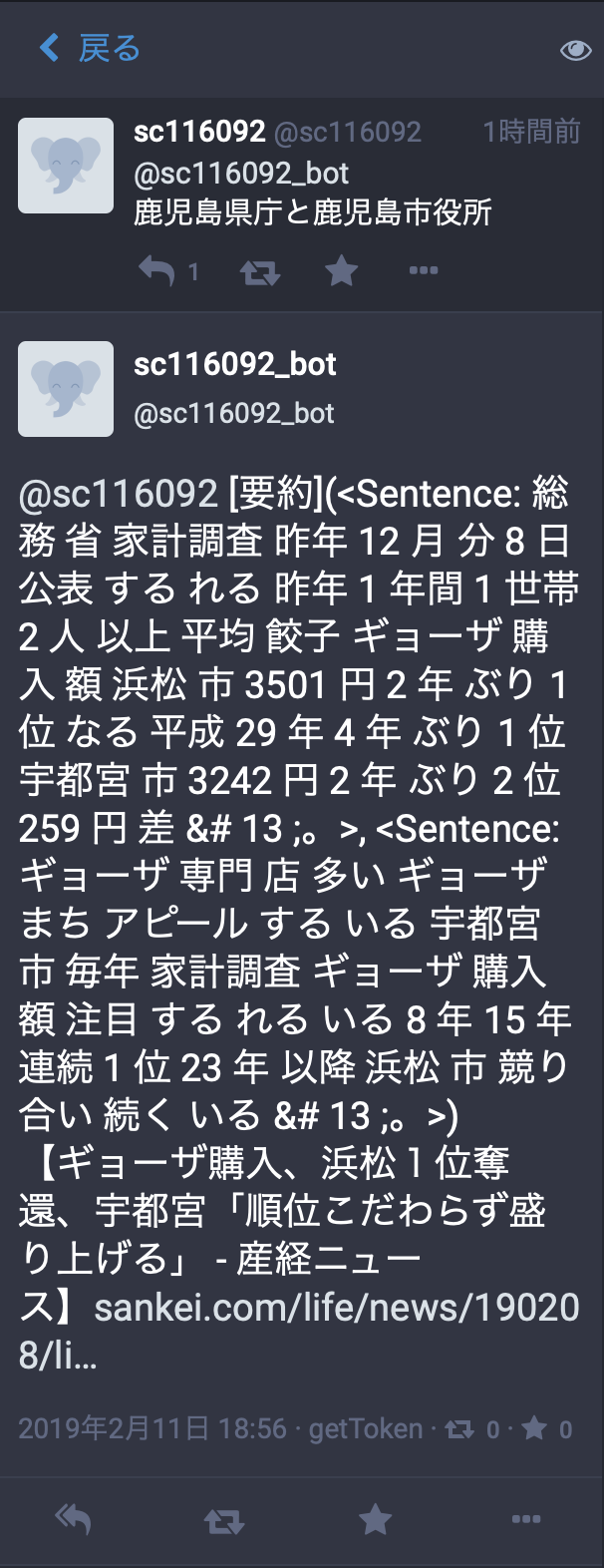
 

図３．実行例３　　　　　　　　　　　図４．実行例４

1. 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．

1. 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．

1. 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）