10 Jan 2020

I. An Assessment of the Propagation of Contemporary Russian Mechanical Phonation Verbs to Online Dictionaries. Web Scraping Project for the Russian National Corpus

Perry B. Koob

Irina V. Ivliyeva
Missouri University of Science and Technology, ivliyeva@mst.edu

Koob, P., Ivliyeva, I. ***An Assessment of the Propagation of Contemporary Russian Mechanical Phonation Verbs to Online Dictionaries. Web Scraping Project for the Russian National Corpus.*** Missouri S&T, IT and ALP departments. [Electronic resource].
Perry B. Koob, koobp@mst.edu; Irina V. Ivliyeva, ivliyeva@mst.edu

This is designed to be run on Windows using Anaconda Jupyter Notebook and Python 3.

2019-04-10
The purpose of this code is to take the list of Russian verbs provided by Dr. Irina Ivliyeva, and for each word in the list, generate a past tense feminine, masculine, neuter, and plural conjugations of the word and then search the Russian National Corpus for the occurances of each word.

2019-11-26 Update:
koobp
Rerunning the script for Dr. Ivilyeva to see if things have changed in the RNC. The URL has changed due to a migration to a new search script. Modifications have also been made to compile results after several runs to handle server initiated disconnects. The old search is still availabe, so I have opted to use that.

2019-12-10 Update:
koobp
Rerunning the script for Dr. Ivilyeva to see if the new and to old application has the same data.

In [ ]:
```
### Import libaries and generate paths to input and output data files

## Import libraries
# Used to read the current working directory
import os
import time
import datetime

import numpy as np
import pandas as pd

# Used to read the contents of a webpage at a given url
import requests

# Used to parse html source
from bs4 import BeautifulSoup
```

Generating the list of words to search

The list of verbs has been provided by Dr. Ivliyeva and converted to a comma separated value file.

The comma separated value has three named columns (number, root, and word). The number is a sequential number that corresponds to the root word, the root word is the root of the word, and the word is the word we are going to search the Russian National Corpus for along with the conjugations.

For purposes of this script, only the word column is important. If you would like to run this for yourself provide a comma separated value file of a list of verbs, and make sure the column header is word. This script is also designed to look for this list of words in a folder called **src-data** in the same folder you ran Jupyter notebooks from.The output of the script will be put in a folder called generated-data which is also located in the folder that Jupyter notebook is run from.

In [ ]:
```
## Read in wordlist from input data file into a data frame.

# Get the working directory.
# This is the directory that jupyter notebook was launched in.  This is assumed
 to be the markdown folder.
cwd = os.path.dirname(os.getcwd())

# Generate path to source data.
srcdatadir = cwd+"\\src-data\\"

# Generate path to generated data.
gendatadir = cwd+"\\generated-data\\"

# Name of the Source file
word_file ="ivliyeva-russian-verbs-extended-corrected.csv"

# The word list that is read in is a utf-8-bom file, that is why the encoding is
set to utf-8-sig.
wordlist = pd.read_csv(srcdatadir+word_file, encoding='utf-8-sig')

# Display the word list.
wordlist.head()
```

The words from the provided list are conjugated using the following method.

Russian Verb Conjugation and Morphology: initial data selection process

By Dr. Irina Ivliyeva

Below I will describe how to modify the infinitives into the past tenses of two verb's nests. The marker of the infinitive is ТЬ at the end of each verb.

The sample group is:

1. Блеять
2. Заблеять
3. Поблеять

The past tense is formed as such:

1. Remove the ть блеять (infitive) to блея- (stem)
2. The past tense in Russian is gender specific, so we have to add the following to the stem:

- Masculine Add л блея+л= блеял
- Feminine Add ла блея+ла= блеяла
- Neuter Add ло блея+ло= блеяло
- Plural Add ли блея+ли= блеяли

The past tense is formed as such:

1. Remove the ть заблеять (infitive) to заблея- (stem)
2. The past tense in Russian is gender specific, so we have to add the following to the stem:

- Masculine Add л заблея+л = заблеял
- Feminine Add ла заблея+ла = заблеяла
- Neuter Add ло заблея+ло = заблеяло
- Plural Add ли заблея+ли = заблеяли

This process is very straitforward and has almost no exceptions. I would like you to run the past tense searches for these 6 verbs:

- блеять (блеял, блеяла, блеяло, блеяли)
- заблеять (заблеял, заблеяла, заблеяло, заблеяли)
- поблеять (поблеял, поблеяла, поблеяло, поблеяли)
- басить (басил, басила,басило, басили)
- забасить (забасил, забасила, забасило, забасили)
- пробасить (пробасил, пробасила, пробасило, пробасили)

There are a few that do not end with "ть" such as "разахаться", that type of verbs called reflexive (-СЯ is a particle equivalent to SELF in English).

The infinitives end on – ТЬСЯ: "разахаться" = "разаха (stem) - ться".

Then, we add the following to make the past tense:

- Masculine разаха +лся = разахался
- Feminine разаха + лась = разахалась

- Neuter разаха + лось = разахалось
- Plural разаха+лись = разахались

In [ ]:
```
## Tense and Conjugation expansion of words
#Word list is generated from the words stripped out of wordlist
words = pd.DataFrame(wordlist.word)
# The initial complete_list is generated
complete_list = words

# Generate the new data frame from the wordlist
df_new = wordlist

#Feminine word forms
tmp = words
tmp = tmp.replace(to_replace='ть$', value='ла', regex = True)
tmp = tmp.replace(to_replace='ться$', value='лась', regex = True)

complete_list = pd.concat([complete_list,tmp],sort=False,ignore_index=True )

tmp.rename(columns={'word':'past_feminine'},inplace=True)
df_new = pd.concat([df_new,tmp],axis=1,sort=False)

#Masculine word forms
tmp = words
tmp = tmp.replace(to_replace='ть$', value='л', regex = True)
tmp = tmp.replace(to_replace='ться$', value='лся', regex = True)

complete_list = pd.concat([complete_list,tmp],sort=False,ignore_index=True )

tmp.rename(columns={'word':'past_masculine'},inplace=True)
df_new = pd.concat([df_new,tmp],axis=1,sort=False)

#Neuter word forms
tmp = words
tmp = tmp.replace(to_replace='ть$', value='ло', regex = True)
tmp = tmp.replace(to_replace='ться$', value='лось', regex = True)

complete_list = pd.concat([complete_list,tmp],sort=False,ignore_index=True )

tmp.rename(columns={'word':'past_neuter'},inplace=True)
df_new = pd.concat([df_new,tmp],axis=1,sort=False)

#Plural word forms
tmp = words
tmp = tmp.replace(to_replace='ть$', value='ли', regex = True)
tmp = tmp.replace(to_replace='ться$', value='лись', regex = True)

complete_list = pd.concat([complete_list,tmp],sort=False,ignore_index=True )

tmp.rename(columns={'word':'past_plural'},inplace=True)
df_new = pd.concat([df_new,tmp],axis=1,sort=False)
```

In [ ]:
```
# Display the complete_list with all the words and conjugations.
complete_list.head()
```

In [ ]:
```
# Display the original word list with the conjugations.
df_new.head()
```

Web scraping the Russian National Corpus for the provided words.

The following process involves a technique called webscaping. Many modern web sites, especailly ones that are associated database application, are generated in an automated fashion. This means that much of the content is wrapped in html formating that is very repatable and predictable. Webscrapping is a technique where you use knowlege of html and the fact the formating is auto-generated, to remove the html formating and extract all the data.

Ideally you would not need to do this, but not all database applications make the process of using their data easy. The ones that do normally provide an Application Programmer Interface (API). The Russian National Corpus does not appear to have an API, so we use the web scraping technique.

In [ ]:
```
# Generate empty lists to be filled with results
corpus_document = list()
corpus_sentences = list()
corpus_words = list()
results_documents = list()
results_context = list()
word_result_text = list()

successfull_words = list()

run_words = complete_list.word
```

The following URLs are not the base URLs used by the Russian National Corpus. They were built using a technique called URL hacking. The URLs below are designed to return more results than the default URL.

In [ ]:
```
# The base Url for the Russian National Corpus (RNC) search
# This URL is for the old search
baseurl = 'http://search1.ruscorpora.ru/search.xml?mycorp=&mysent=&mysize=1000&dpp=1000&spp=1000&spd=10&t=1000&text=lexform&mode=main&sort=gr_tagging&lang=en&req='
# This URL is for the new search
baseurl = 'http://search.ruscorpora.ru/search.xml?mycorp=&mysent=&mysize=1000&dpp=1000&spp=1000&spd=10&t=1000&text=lexform&mode=main&sort=gr_tagging&lang=en&req='
```

In [ ]:
```
start_time = time.time()

run_words = [x for x in complete_list.word if x not in successfull_words]

# For each word in the complete word list
for word in run_words:

    # Generate a seach url by concating the word to be searche to the end of the
baseurl
    workingurl = baseurl + word
    # Use the working URL to query the RNC and get the html for the result page
    page = requests.get(workingurl).text

    if requests:
        # Load the html into BeautifulSoup
        soup = BeautifulSoup(page, 'lxml')

        # Get all the html nodes that have class=stat-number
        # Html nodes that are styled by the class stat-number
        # have various number stats for the corpus and the results.
        # These numbers are in a format where a space is used instead of a comm
a.
        # If no results were returned, there are no html nodes styled by stat-nu
mber
        stat_number =  soup(attrs={'class': 'stat-number'})

        # Determine if results were returned
        if len(stat_number) > 0:
            # If classes were found, then the string is converted to a number
            # and added to the appropriate list.
            corpus_document.append((stat_number[0].get_text()).replace(' ',''))
            corpus_sentences.append((stat_number[1].get_text()).replace(' ',''))
            corpus_words.append((stat_number[2].get_text()).replace(' ',''))
            results_documents.append((stat_number[3].get_text()).replace(' ',''
))
            results_context.append((stat_number[4].get_text()).replace(' ',''))
        else:
            #If no classes were found, then 'NA' was added to the appropriate li
st.
            corpus_document.append('NA')
            corpus_sentences.append('NA')
            corpus_words.append('NA')
            results_documents.append('NA')
            results_context.append('NA')

        # Get all the html list nodes
        # List nodes have been determined to be used for results only.
        # If no results were returned, there are no li nodes
        results = soup.find_all('li')
        result_text = ''


        if len(results) > 0:
            # If li nodes were found, then every other one is processed.
            # There appears to be some duplication.
            for i in range(0,len(results),2):
                line = results[i].get_text()
                # Strip out all tabs
```

```
                line = line.replace('\t','')
                # Strip out all of these strange combinations of characters at t
he end of each
                # results context text.
                line = line.replace('←…→','')
                # Strip out all of these large number of spaces that proceed eac
h result context text.
                line = line.replace('           ','')
                # Add two newlines to the end of each results context text.
                line = line + '\n\n'
                result_text = result_text + line
        else:
            #If no li nodes were found, then 'NA' was added to the appropriate l
ist.
            result_text = 'NA'

        # Remove any carraige returns
        result_text = result_text.replace('\r','')

        # Remove any double newline at the end of the collected results text
        result_text = result_text.replace('\n\n$','')

        # Append the results to the appropriate list.
        word_result_text.append(result_text)

        # Add to successfull word list
        successfull_words.append(word)

        # Status output
        if (len(successfull_words) % 50) == 0:
            print(round(time.time() - start_time,2),len(successfull_words),'of',
len(complete_list.word))

len(successfull_words)
```

After the Russian National Corpus has been web scraped, frequency counts are compile and written to Excel.

In [ ]:
```
corpus_document
```

In [ ]:
```
# Append count of corpus document count to the complete list.
cd = pd.DataFrame(corpus_document)
cd.rename(columns={0:'corpus_document'},inplace=True)
complete_list = pd.concat([complete_list,cd],sort=False, axis=1)

# Append count of corpus sentence count to the complete list.
cs = pd.DataFrame(corpus_sentences)
cs.rename(columns={0:'corpus_sentences'},inplace=True)
complete_list = pd.concat([complete_list,cs],sort=False, axis=1)

# Append count of corpus word count to the complete list.
cw = pd.DataFrame(corpus_words)
cw.rename(columns={0:'corpus_words'},inplace=True)
complete_list = pd.concat([complete_list,cw],sort=False,axis=1)

# Append count of corpus results document count to the complete list.
rd = pd.DataFrame(results_documents)
rd.rename(columns={0:'results_documents'},inplace=True)
complete_list = pd.concat([complete_list,rd],sort=False,axis=1)

# Append count of corpus results context count to the complete list.
rc = pd.DataFrame(results_context)
rc.rename(columns={0:'results_context'},inplace=True)
complete_list = pd.concat([complete_list,rc],sort=False,axis=1)

# Append count of results context text to the complete list.
wrt = pd.DataFrame(word_result_text)
wrt.rename(columns={0:'results'},inplace=True)
complete_list = pd.concat([complete_list,wrt],sort=False,axis=1)
```

In [ ]:
```
# Get date stamps for filenames
date = datetime.datetime.now()
datestamp = str(date.year) + '-' + str(date.month).rjust(2,'0') + '-' + str(date
.day)
```

In [ ]:
```
# Write the complete_list out as an Excel file

out_file = "rnc-text-mining-conjugations-" + datestamp + ".xlsx"
df_new.to_excel(gendatadir + out_file, index=False)
```

In [ ]:
```
# Display the completed complete_list
complete_list.head()
```

In [ ]:
```
# Write the complete_list out as an Excel file
out_file = "rnc-text-mining-output-" + datestamp + ".xlsx"
complete_list.to_excel(gendatadir + out_file)
```

In [ ]:
```
# Reform the frequency to generate frequency matrix for returned documents that
 is paired with words and conjugations
type(rd)
rd
columns = 5
rows = int(len(rd)/columns)
matr = rd.values
df_doc_fm = pd.DataFrame(np.reshape(matr,(columns,rows)).transpose())
df_doc_fm.rename(columns={0:'word_dfreq',1:'past_feminine_dfreq',2:'past_masculi
ne_dfeq',3:'past_neuter_dfeq',4:'past_plural_dfeq'},inplace=True)
#df_new
df_doc_fm
complete_doc_frequency = pd.concat([df_new,df_doc_fm],sort=False,axis=1)
```

In [ ]:
```
complete_doc_frequency.head()
```

In [ ]:
```
# Write the complete_doc_frequency out as an Excel file
out_file = "rnc-text-mining-doc-frequency-" + datestamp + ".xlsx"
complete_doc_frequency.to_excel(gendatadir + out_file)
```

In [ ]:
```
# Reform the frequency to generate frequency matrix for returned contenxts that
 is paired with words and conjugations
type(rc)
rd
columns = 5
rows = int(len(rc)/columns)
matr = rc.values
df_con_fm = pd.DataFrame(np.reshape(matr,(columns,rows)).transpose())
df_con_fm.rename(columns={0:'word_cfreq',1:'past_feminine_cfreq',2:'past_masculi
ne_cfeq',3:'past_neuter_cfeq',4:'past_plural_dceq'},inplace=True)
#df_new
df_con_fm
complete_con_frequency = pd.concat([df_new,df_doc_fm],sort=False,axis=1)
```

In [ ]:
```
complete_con_frequency.head()
```

In [ ]:
```
# Write the complete_doc_frequency out as an Excel file
out_file = "rnc-text-mining-con-frequency-" + datestamp + ".xlsx"
complete_con_frequency.to_excel(gendatadir + out_file)
```

In [ ]:
```
import dill
```

In [ ]:
```
dill.dump_session(gendatadir+"notebook_env-"+ datestamp +".db")
```