ISSN: 2350-0328
International Journal of Advanced Research in Science, Engineering and Technology
Vol. 6, Issue 12 , December 2019

The basics of processing Uzbek text using Python

Abdisait Norov, Dilfuza Saidova, MohidilSayrikhonova

P.G. Student, Department of Methods of teaching Informatics, Karshi State University, Karshi, Uzbekistan
P.G. Student, Department of Methods of teaching Informatics, Karshi State University, Karshi, Uzbekistan
Student, Faculty of physics and mathematics, Karshi State University, Karshi, Uzbekistan

ABSTRACT: Currently, Python has several applications: Machine Learning, Natural Language Processing, Neural Networks, Data analysis, Web-development, etc. It has many modules and a sufficient number of libraries focused on Natural Language Text Processing. In this article, the author analyzes the methods of using Python and its NLTK library for processing Uzbek texts, in particular, the definition of Stop Words, tokenization of words and sentences in the Uzbek text.

KEY WORDS: linguistic model; software technologies; Python; NLTK; Stop-words; text corpus.

I. INTRODUCTION

Currently, there are a number of software technologies for natural language research. These include various natural language processing technologies, such as programming systems, various software libraries with linguistic models, linguistic processors, linguistic databases, morphological, semantic and syntactic analyzers,and etc.

Recent research on automatic linguistic data processing shows that Python is at the forefront of its simplest instructions and huge libraries among many programming systems such as Java, Perl, C++, C#, Ruby, Visual Basic, R, and etc.

Currently, a large number of libraries for the analysis of natural language texts containing sets of basic algorithms have been developed. For example, Python and its packages (SpaCy, Gensim NLTK, etc.) are currently used in most regions of the world as natural language text processing methods. And the most famous package of tools for the Russian language in this area is AOT, RussianMorphology, RussianPOSTagger, mystem, and etc.

However, it can be emphasized that for the Uzbek language similar software packages for text processing have not yet been found. Although preliminary studies have been carried out on modeling part of speech on the basis of the syntactic structure of the Uzbek language and improving the linguistic foundations for morphological analysis of words, but the creation of various language technologies for the study of the Uzbek language remains as urgent tasks.

II. SIGNIFICANCE OF THE SYSTEM

The paper mainly focuses on how use Python libraries in Processing of Uzbek Text. The study of literature survey is presented in section III, Methodology is explained in section IV, section V covers the experimental results of the study, and section VI discusses the future study and Conclusion.

III. LITERATURE SURVEY

In general, there is not much literature devoted to the processing of the Uzbek language on the basis of software technologies. However, some of the methods described in the literature can be used as examples.

Большакова Е.И. и др.The book discusses the basic issues of computer linguistics: from the theory of linguistic and mathematical modeling to options for technological solutions. The linguistic interpretation of the main linguistic objects and units of analysis is givenfor Russian text.The methods described in this book can be used in the study of Processing Uzbektext.

Хакимов M.This monograph considers the technology of a multilingual simulated computer translator, from the point of view theoretically for Uzbek texts.

IV. METHODOLOGY

When automatically processing natural languages, it uses different methods for different languages, but the following methods are common to most natural languages:
1. Sentence tokenization;
2. Word tokenization;
3. Text lemmatization and stemming;
4. Stop-words;
5. Regular expressions;
6. Bag-of-Words;
7. TF-IDF

Sentence tokenization (also called sentence segmentation) is the problem of dividing a string of written language into its component sentences.

Word tokenization (also called word segmentation) is the problem of dividing a string of written language into its component words. In many languages using some form of Latin and Cyrillic alphabet, space is a good approximation of a word divider.

For grammatical reasons, documents can contain different forms of Uzbek word such as “bormoq”, “bormoqda”, and “bormoqchi”.The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stop-words are words which are filtered out before or after processing of text. When applying machine learning to text, these words can add a lot of noise. That’s why we want to remove these irrelevant words. Stop-words usually refer to the most common words such as “ва”, “ҳам”, “мана” in a language, but there is no single universal list of stop-words. The list of the stop words can change depending on your application.

The results of our research in this area show that some of the above modules of the NLTK (Natural Language Tool-Kit) library can also be effectively used in modeling elements of Uzbek grammar. Of course, this process cannot be done directly. This requires computer modeling of the relevant objects of the Uzbek language.

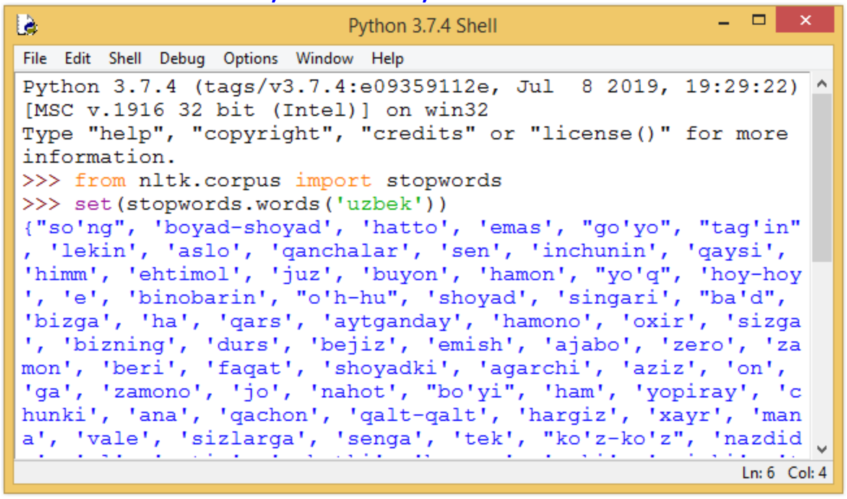
V. EXPERIMENTAL RESULTS

As an example, below is the source code for screening Uzbek stop-words, created by the author as a linguistic database using the module " NLTK.Corpus” [7], [8]:

```
>>> import nltk
>>> nltk download()
>>> from nltk.corpus import stopwords
>>>set(stopwords.words('uzbek'))
```

The result is as follows (Fig 1):

ISSN: 2350-0328
International Journal of Advanced Research in Science, Engineering and Technology
Vol. 6, Issue 12 , December 2019



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22)
[MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> from nltk.corpus import stopwords
>>> set(stopwords.words('uzbek'))
{"so'ng", 'boyad-shoyad', 'hatto', 'emas', "go'yo", "tag'in"
, 'lekin', 'aslo', 'qanchalar', 'sen', 'inchunin', 'qaysi',
'himm', 'ehtimol', 'juz', 'buyon', 'hamon', "yo'q", 'hoy-hoy
', 'e', 'binobarin', "o'h-hu", 'shoyad', 'singari', "ba'd",
'bizga', 'ha', 'qars', 'aytganday', 'hamono', 'oxir', 'sizga
', 'bizning', 'durs', 'bejiz', 'emish', 'ajabo', 'zero', 'za
mon', 'beri', 'faqat', 'shoyadki', 'agarchi', 'aziz', 'on',
'ga', 'zamono', 'jo', 'nahot', "bo'yi", 'ham', 'yopiray', 'c
hunki', 'ana', 'qachon', 'qalt-qalt', 'hargiz', 'xayr', 'man
a', 'vale', 'sizlarga', 'senga', 'tek', "ko'z-ko'z", 'nazdid
```

Fig 1: Screening of Uzbek words in the Stop-words type

We can also use the NLTK package tokenization module (*nltk.tokenize*) for Uzbek words and sentences. Let's say we have the following text:

“*Men dasturchiman. Mendasturlashgajuda qiziqaman*.”
(*I am a programmer. I am very interested in programming*).

Each sentence, word and punctuation in this Uzbek text can be divided into separate blocks. Initially, we present the whole text of the program on tokenization of words:

```
>>>import nltk
>>> from nltk.tokenize import word_tokenize
>>>MyText="Men dasturchiman. Mendasturlashgajuda qiziqaman."
>>> print(word_tokenize(MyText))
```

The result of the program:

```
['Men', 'dasturchiman', '.', 'Men', 'dasturlashga', 'juda', 'qiziqaman', '.']
```

Now let's give the text of the program on tokenization of proposals.

```
>>> import nltk
>>> from nltk.tokenize import sent_tokenize
>>>MyText="Men dasturchiman. Mendasturlashgajuda qiziqaman."
>>>print(sent_tokenize(MyText))
```

The result of the program:

```
[Men dasturchiman.', 'Mendasturlashgajuda qiziqaman.']
```

One problem with scoring word frequency is that the most frequent words in the document start to have the highest scores. These frequent words may not contain as much “informational gain” to the model compared with some rarer and domain-specific words. One approach to fix that problem is to penalize words that are frequent across all the documents. This approach is called TF-IDF.

TF-IDF short for term frequency-inverse document frequency is a statistical measure used to evaluate the importance of a word to a document in a collection or text corpus. The formula used to calculate a TF-IDF score for a given term ***x*** within a document ***y*** is:

ISSN: 2350-0328
International Journal of Advanced Research in Science, Engineering and Technology
Vol. 6, Issue 12 , December 2019

$$W_{x,y} = TF_{x,y} \times \log\left(\frac{N}{DF_x}\right)$$

TF is the frequency of a term that measures how often a term appears in a document. It is logical to assume that in long documents the term may occur in larger quantities than in short ones, therefore absolute numbers do not roll here. Therefore, relative ones are used – they divide the number of times when the desired term is encountered in the text by the total number of words in the text. And so, formula of TF:

$$TF\ of\ the\ term\ \boldsymbol{a} = \frac{The\ number\ of\ times\ the\ term\ \boldsymbol{a}\ occurred\ in\ the\ text}{The\ number\ of\ all\ words\ in\ the\ text}$$

In code form, it will look like this (as example of Uzbek text):

```
>>> import collections
>>>def compute_tf (text):
  tf_text = collections.Counter (text)
   for i in tf_text:
       tf_text [i] = tf_text [i] / float (len (text))
   return tf_text

>>>text = ['men', 'dasturchiman', 'men', 'dasturlashga', 'juda', 'qiziqaman']
>>>print(compute_tf (text))
```

The result of the program:

```
Counter({'men': 0.3333333333333333, 'dasturchiman': 0.16666666666666666, 'dasturlashga': 
0.16666666666666666, 'juda': 0.16666666666666666, 'qiziqaman': 0.16666666666666666})
```

IDF is the inverse frequency of documents. It directly measures the importance of the term. That is, when we considered TF, all terms are considered as if equal in importance to each other. But everyone knows that, for example, prepositions are very common, although they practically do not affect the meaning of the text. And what to do about it? The answer is simple – count IDF. It is considered as the logarithm of the total number of documents divided by the number of documents in which the term “a” is found. And so, formula of IDF:

$$IDF\ term\ \boldsymbol{a} = log\left(\frac{total\ number\ of\ papers}{number\ of\ documents\ in\ which\ the\ term\ occurs\ \boldsymbol{a}}\right)$$

The logarithm, by the way, can be taken by anyone – because TF-IDF is a relative measure; that is, the weights of the terms are not expressed in some units, but exist relative to each other. For example, I usually take natural or decimal.

In code form, it will look like this (as example of Uzbek text):

```
>>>import math
>>>def compute_idf (word, corpus):
      return math.log10 (len(corpus)/sum([1.0 for i in corpus if word in i]))
>>>texts = [['men', 'dasturchiman'], ['men', 'dasturlashga', 'juda', 'qiziqaman']]
>>>print('IDF("juda"): ', compute_idf ('juda', texts))
```

The result of the program:

```
IDF("juda"): 0.3010299956639812.
```

To calculate TFI×DF multiplication, the code is as follows:

```
>>>from collections import Counter
>>>import math
>>>def compute_tfidf(corpus):
  def compute_tf(text):
      tf_text = Counter(text)
     for i in tf_text:
        tf_text[i] = tf_text[i]/float(len(text))
      return tf_text

  def compute_idf(word, corpus):
      return math.log10(len(corpus)/sum([1.0 for i in corpus if word in i]))
 documents_list = []
  for text in corpus:
       tf_idf_dictionary = {}
       computed_tf = compute_tf(text)
     for word in computed_tf:
         tf_idf_dictionary[word] = computed_tf[word] * compute_idf(word, corpus)
        documents_list.append(tf_idf_dictionary)
  return documents_list

>>>corpus = [['men', 'dasturchiman'],
        ['men', 'dasturlashga', 'juda', 'qiziqaman']]
>>>print(compute_tfidf(corpus))
```

The result of the program:

```
[{'men': 0.0, 'dasturchiman': 0.1505149978319906}, {'men': 0.0, 'dasturlashga':
0.0752574989159953, 'juda': 0.0752574989159953, 'qiziqaman': 0.0752574989159953}]
```

The lack of modern technology of analysis and synthesis of texts of the Uzbek language hinders further research in the field of processing of the Uzbek text, in particular, in the field of graphematical, morphological and semantic-syntactic analysis, and etc.

It is known that the Uzbek language belongs to the Turkic family of languages (languages that form a number of groups, which include the languages Turkish, Azerbaijani, Kazakh, Kyrgyz, Turkmen, Uzbek, Karakalpak, Uighur, Tatar, Bashkir, Chuvash, Balkar, Tuvan, etc.) and belongs to the class of agglutinative languages. And for agglutinative languages, it is characteristic that the successive accession of various formative endings. For example:

Maktab (School)
Maktablar (Schools)
Maktablarimiz (Our schools)
Maktablarimizdagilar (At our schools)
Maktablarimizdagilarning (Of our schools)

We can distinguish affixes from a given last word-form:

Maktab + lar + i + miz + da + gi + lar + ning.

For this reason, this situation, of course, complicates the process of processing the Uzbek language than other inflectional languages.

VI.CONCLUSION AND FUTURE WORK

The above-mentioned software technologies and all linguistic models studied with their help are actually the basis for the machine translation and machine learning procedure. We need a lot of formal models for studying various aspects of the Uzbek language with the use of these technologies. The research in this article can serve as a basis for creating a software package for processing Uzbek text. In the future, we study other functions of Python, in a study of Uzbek texts.

REFERENCES

[1]. Bolshakova E.I. and others.Automatic processing of natural language texts and Computer Linguistics. – Moscow, 2011. – 272 p.
[2]. Khakimov M. Multilanguage Simulated Computer Translator Technology. Monograph //Riga, “LAPLAMBERTAcademicPublishing”, 2019. – 174 p.