International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 06 Issue 2
February 2019

NLP ALGORITHMS IN OOP FOR PROCESSING THE TEXT BASED DOCUMENTS IN RUSSIAN LANGUAGE IN MACHINE LEARNING.

N. Urmanov[1], Bektemyssova G.U[2], Kouros Basiri[3]
[1]International University of Informational Technology, 050040,Almaty, Kazakhstan
urmnurbolat@gmail.com
[2]International University of Informational Technology, 050040,Almaty, Kazakhstan
k.basiri@iitu.kz
[3]International University of Informational Technology, 050040,Almaty, Kazakhstan
g.bektemysova@iitu.kz

ABSTRACT

The purpose of this article is to guide in natural language processing algorithms in Object-Oriented Programming for processing Russian text documents in machine learning. In this work, we describe some NLP algorithms and tools, and how to apply and use best practice Object-Oriented Programming for refine Russian text in machine learning.

KEYWORDS

Natural language processing; sentiment analysis; machine learning; tokenization; word frequency; tf-idf; Word2Vec; stemming; lemmatization.

1. INTRODUCTION

In every second the volume of data in worldwide increase significantly, which consist information such as photos, videos, comments in social networks, reviews in internet shops, articles, books and etc.

Human physically cannot process and monitor such capacity of information. For example, nowadays in Facebook work around 7 500 content moderators, to review information which is located in one social network. In addition, some kind of information consist forbidden content, which can be harmful to psychology of the person.

Therefore was developed set of algorithms which applied by computer which allows to process large volume of information.

Natural language processing (Natural Language Processing, NLP) - the general direction of an artificial intelligence, mathematical linguistics and machine learning. It studies problems of the computer analysis and synthesis of natural languages.

The purpose of NLP is processing and "understanding" of a natural language for the translation of the text to machine language and the response to questions.

The quality of understanding depends on a set of factors: from language, from national culture, from the interlocutor etc.

The main directions of natural language processing include such as extraction of the facts, sentiment analysis, responses to questions, information search, generation of the text, translation, etc. In this article, we will cover algorithms in Object-oriented language, which applied in sentiment analysis in Russian language.

2. Natural language processing – algorithms and tools

Semantic analysis - extremely heavy operation from the point of view of use by developers: most often the deep understanding of the algorithm and methods of its implementation and also knowledge of the language on which the text is written is required.

NLP in semantic analysis include next algorithms, which covered in this work:

- Tokenization
- Word frequency in text
- Stemming
- Stop-listing
- Lemmatization
- TF-IDF matrix
- Word2Vec

Those algorithms of NLP is common and widely used in semantic analysis.

2.1. Tokenization

Token - the object which is created from a lexeme in the course of lexical analysis. Token in lexeme is the minimum unit of language making independent sense. There are following types of lexemes. In other words, in semantic analysis token can be single word, sentences or paragraph. In this work we will use token as a single word.

The first step in NLP is to identify tokens, or those basic units which need not be decomposed in a subsequent processing. The entity word is one kind of token for NLP, the most basic one. Our concern, however, is with using the computer to recognize those tokens without distinct delimiters, such as Chinese words, English idioms and fixed expressions [4]

2.2. Word frequency in text

In semantic analysis often use word counter algorithm, to find frequency of each word in given text. Why this technique need? It need if your task is find keywords or unique terms, to classify large documents corpus, to find corpus similarity and etc.

The distribution of word frequencies is a fundamental phenotype of a language. Statisticians and linguists have studied Word frequency distributions since the statistics of word usage yield valuable insights into the language, its construction, and its evolution. These distributions have been long-studied outside of statistics and linguistics as well. [3]

2.3. Transformed Term-Frequency Matrix

The entries in the term-document matrix are often transformed to weight them by their estimated importance in order to better mimic the human comprehension process. For language simulation, the best performance is observed when frequencies are cumulated in a sublinear fashion within cells (typically log(freqij+1), where freqij is the frequency of term i in document j), and inversely with the overall occurrence of the term in the

collection (typically using inverse document frequency or entropy measures). (Figure 1) [1]

	Docs																			
Terms	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
data	1	1	0	0	2	0	0	0	0	0	1	2	1	1	1	0	1	0	0	0
examples	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
introduction	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
mining	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0
network	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
package	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 1 Term-by-document matrix.

2.4. Stop-listing and stemming

In stemming, conversion of morphological forms of a word to its stem is done assuming each one is semantically related. The stem need not be an existing word in the dictionary but all its variants should map to this form after the stemming has been completed. There are two points to be considered while using a stemmer:

- Morphological forms of a word are assumed to have the same base meaning and hence should be mapped to the same stem

- Words that do not have the same meaning should be kept separate

International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 06 Issue 2
February 2019

These two rules are good enough as long as the resultant stems are useful for our text mining or language processing applications. Stemming is generally considered as a recall-enhancing device. For languages with relatively simple morphology, the influence of stemming is less than for those with a more complex morphology. Most of the stemming experiments done so far are for English and other west European languages.[2]

Stop-listing is applied to delete stop words and single symbols which is appear in text document, and probably frequency of stop words can be more than other key words in this text. That's why this step is necessary to use for get more clear result in machine learning.

2.5. Lemmatization

Lemmatization is transformation of words into a lemma, that is, into their original vocabulary form. First of all, lemmatization is used by search engines. It helps them to accelerate indexing and request processing and also to increase relevance of the delivery. This is exact process with use of a lexicon and morphological analysis of words as a result of which only the inflectional terminations are removed and returns the main, or dictionary, the word form called by a lemma. For example, in "saw" lexeme during stemming can turn into letter "s" while lemmatization will return either the word "see", or the word "saw "depending on whether the lexeme is a verb or a noun. The important distinction consists that usually stemming "sticks together" derivative cognate words, and lemmatization "sticks together" only inflected forms of one lemma. Stemming and lemmatization are often carried out by means of the additional program components which are built in indexing process.

As proposed in, each word is labeled by a class label, that represents the transformation that should be applied to get the normalized form of the word. To determine this class, a stem should be found first. It is the part the two words (the word and its normalized form) have in common. The words property and properties have both the stem "propert" in common. Or in Slovene, the words "BRESKEV" and "BRESKVAH" have "BRESK" in common. Then we can see that we should remove the suffix "VAH" from "BRESKVAH" and add the suffix "EV" to get the normalized form "BRESKEV". Thus we assign the class label "VAHtoEV" to the word "BRESKVAH" [5]

2.6. Word2Vec

Word2Vec is a tool (a set of algorithms) for calculation of vector representations of words, implements two main architecture — Continuous Bag of Words (CBOW) and Skip-gram. As a input define text corpus or word, and output will be represent as vector variables (coordinators in vector space).

Continuous Bag of Words (CBOW) and Skip-gram illustrated in Figure 2.

The skip-gram model's objective function is to maximize the likelihood of the prediction of contextual words given the center word. More formally, given a document of T words, we wish to maximize (1)

$$L = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\neq 0} \log\left(w_{t+j} \mid w_t\right) \quad (1)$$

Where c is a hyperparameter defining the window of context words. [6]

The CBOW model predicts the center word w_o given a representation of the surrounding words w_{-c}, ..., w_{-1}, w_1, w_c. Thus, the output vector $o_{w-c},\ldots,w_{-1},w_1,w_c$ is obtained from the product of the matrix $O \in R^{|V|*d_w}$ with the sum of the embeddings of the context words (2) [6]

$$\sum_{-c\le j\le c, j\neq 0} r_{w_j} \quad (2)$$

International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 06 Issue 2
February 2019

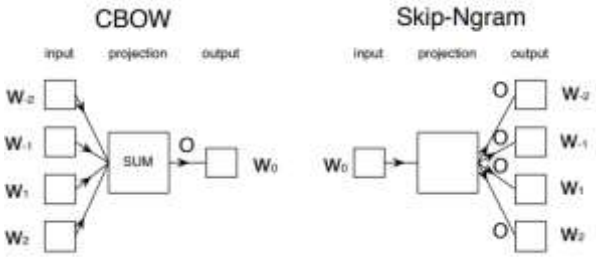

Figure 2. Skip-gram and Continuous Bag-of-Word (CBOW) models illustration

The principle of set of algorithms is finding of relations between contexts of words according to the assumption that the words that are in similar contexts tend to mean similar things, i.e. to be semantic close. More formally: maximizing cosine proximity between vectors of words (a scalar product of vectors) which appear next to each other, and minimization of cosine proximity between vectors of words which do not appear the friend near the friend. Next to each other in this case means in familiar contexts.

Word2Vec can be applied perfectly to different problems of natural language processing, somehow:

- Word clustering by the principle of their semantic similarity
- Identification of semantic similarity of words (for example to what a word the cat — is semantic closer to food or space pirates)
- For the analysis of tonality, however, not successfully. That is, quite successfully, but after all not really.

3. Experimental part

In this part we will describe how to apply mentioned above algorism for Russian language. It will include some description, pseudocode and examples in object-oriented programming.

3.1. Applying Tokenization

In object oriented programming many example of how to apply tokenization in text input. In this work, we will use technique that is applicable for machine learning and can be reusable.

The basic of tokenization is to use Iterator<Template> object, which is available in java.util library.In text-based document Template should be as a String. In machine learning generally, take as input whole file, which is text-based document, which converted to streamed object. For example, in java language input file converted to BufferReader.

To convert file to Iterator, which can be reusable in program, you should apply next algorithm:

1. Read text line by line. In each OOP language document can be read line by line by function readline().

2. Divide each object in line by whitespacing.

Each object as string should be saved in Iterator<Template> object, where store each word, symbol or a single char. After this we can apply different algorithms, like stemming, word counting, lemmatization to each single object, which stored in Iterator<Template> object. Tokenization pseudocode in Algorithm 1.

Algorithm 1: Tokenization Algorithm

1. t:=f(u,X)//[X¯]n2
2. c:=E(K,t)
3. if (c¯mod2n)≥10ℓ, then t:=c and go back to step 2
4. token:=[c¯mod2n]ℓ10

5. if check(token)=True, then u:=u+1 and go back to step 1
6. return token

3.2. Applying stop listing

This technique very easy to apply, but very important step in natural language processing. This step important, because it delete noises in machine learning and improve result of learning or analysis.

One if example of using stop listing is to download stop words list file which is available in internet for free and for every language and compare each word in text to each word in stop words list. If they are equal, just ignore these noises. Each word can be retrieved from tokenization, which is described above. Example of usage in OOP code in Example 1.

Example 1: Stop listing usage example

```
If (Stop Words List.contains(token))
        Skip;
```

3.3. Applying stemming

The problem of text analysis, is that there are appear one word with different affix. For example in English: user – 3 times, users – 4 times. In other word NLP will consider this word as a different. To obtain this situation we use stemming algorithms.

To write own stemming algorithm very hard. It needs full morphological analysis of language. The easiest way is to use available tools. In Russian language available very strong tool Snowball stemming. It include methods and algorithms, which will cut Russian endings. For you it only need declare Snowball Russian stemmer object and send as argument each token, which means each word. Example of usage in OOP code in Example 2.

Example 2: Stemming algorithm example

```
RussianStemmer stemmer  = new RussianStemmer();
stemmer.setCurrent(token);
stemmer.stem();
```

Pseudocode for stemming “word” string in Algorithm 2.

Algorithm 2: Stemming Algorithm

```
Input: a text token and a dictionary
Rules:
If token length < 4 return token
If token is number return token
If token is acronym return token
If token in dictionary return the stored stem
If token ends in s’
  strip the ’ and return stripped token
If token ends in ’s
  strip the ’s and return stripped token
```

```
If token ends in “is”, “us”, or “ss” return token
If token ends in s
   strip s, check in dictionary, and return stripped token if there
If token ends with es
   strip es, check in dictionary, and return stripped token if there
If token ends in ies
   replace ies by y and return changed token
If token ends in s
   strip s and return stripped token
If token doesn’t end with ed or ing return token
If token ends with ed
   strip ed, check in dictionary and return stripped token if there
If token ends in ied
   replace ied by y and return changed token
If token ends in eed
remove d and return stripped token if in dictionary
If token ends with ing
   strip ing (if length > 5) and return stripped token if in dictionary
If token ends with ing and length ≤ 5 return token
// Now we have SS, the stripped stem, without ed or ing and it’s
// not in the dictionary (otherwise algorithm would terminate)
If SS ends in doubling consonant
   strip final consonant and return the changed SS if in dictionary
If doubling consonant was l return original SS
If no doubled consonants in SS
   add e and return changed SS if in dictionary
If SS ends in c or z, or there is a g or l before the final doubling consonant
   add e and return changed SS
If SS ends in any consonant that is preceded by a single vowel
   add e and return changed SS
return SS
```

Note that in Russian language uses UTF-8 encoding for cutting endings.

Disadvantages of stemming algorithm is not all the words can be stemmed to their right root word, which leads to not properly natural language processing and less accurate machine learning result.

3.3. Applying lemmatization

Lemmatization is moderate form of stemming. This algorithm can find right roots of word than stemming algorithm. For example: am, are and is after lemmatization converts to be.

To write own lemmatization algorithm will be very hard, harder than stemming. Because lemmatization need some lexical database, which need large of time to create own database. In other word for use lemmatization in machine learning and not spent time, is to use available API. In Russian language there are available SpaCy tools or info.semanticanalyzer library. To use this tools is similar to stemming, send as argument each token. Example of OOP code in Example 3.

Example 3: Lemmatization algorithm use example

```
MorphAnalyzer analyzer = MorphAnalyzerLoader.load(new MorphAnalyzerConfig(New
Properties()));
MorphDesc morphDescription =analyzer.analyzeBest(token);
morphDescription.getLemma();
```

Pseudocode of lemmatization in Algorithm 3.

Algorithm 3: Lemmatization Algorithm

```
1. check the presence of X in the list of lemmas;
     if "yes" then return X itself and stop
2. check the presence of X in the list of guides;
     if "yes" then return the lemma of guide X and stop
3. repeat the following:
   3.1. look for a guide Y with the same ending as the ending of      X (as long as it can be)
   3.2. derive the (probably) lemma X'' of X by comparing with the (known)
        lemma of Y (details beneath)
   3.3. check the presence of X'' in the list of lemmas;
        if "yes" then return X'' and stop
```

3.3. Applying word frequency in text

Word frequency is just counting how many times word appears in text. It seems quite easy, but in NLP, which applied in machine, should be complex and reusable. In other word frequency number of each word should stored in object for further processing.

Word counting should be represented as Map variable, which stored as a key String word, and value as Integer – number of frequency, in Example 4.

Example 4: Word frequency object example

```
Map<String, Integer>
```

OOP code example in Example 5.

Example 5: Word frequency usage example

```
public int count(T obj, int count) {
   if (count < 1)
      throw new IllegalArgumentException("Count must be positive: " + count);
   int objIndex = (allowNewIndices)
      ? objectIndices.index(obj)
      : objectIndices.find(obj);
   if (objIndex < 0)
      return 0;
   int curCount = indexToCount.get(objIndex);
   indexToCount.put(objIndex, curCount + count);
   sum += count;
```

```
    return curCount + count;
  }
```

Pseudocode in Algorithm 4.

Algorithm 4: Word frequency counting Algorithm

```
void Map (key, value){
//key – word
//value – word frequency
      get key index;
     if index not find
      create index;
     value+1;
}
```

3.3. Applying TF-IDF

TF-IDF is statistics which are used mainly for estimation of importance (weightiness) of a concrete word (term) in the context of all document entering the general collection (base)

There are 2 method should be applied:

Term-Frequency(TF) - the relation of number of occurrence of the concrete term to a total mere verbiage in the studied text (document). To show this reflects importance (weightiness) of a word within certain article / publication - tf(t,d) = word frequency/total number of word in document

Inverse document frequency (IDF) - it is inversion of rate with which a certain word appears in a collection of texts (documents). Thanks to this indicator it is possible to reduce weightiness of the most widely used words (pretexts, the unions, the general terms and concepts). For each term within a certain base of texts only the unique IDF value is provided - idf(t,d) = log(word frequency/total number of document containing word)

For computing TF, we need word frequency and number of words in documents. This information can retrieve from word frequency, which described above. Pseudocode of algorithm is in Algorithm 5.

Algorithm 5: Term-frequency Algorithm

```
ComputeTF(Map<key, Value>){
     tf = value/Map.length;
    return tf;
}
```

Example of code in OOP in Example 6.

Example 6: Term-frequency example

International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 06 Issue 2
February 2019

```
ComputeTF (MAP[key, value] of Object.entries(TFVals)){
      TFVals = TFVals[value] / Map.length;
 }
```

IDF computing additional need number if documents which contain key (word). Pseudocode in Algorithm 6.

Algorithm 6: Inverse document frequency Algorithm

```
document count = 0;
ComputeIDF(Map<key, Value>){
      idf = log(value/ ComputeDocuments());
Return idf;
}

ComputeDocuments(Map<key, Value>,documents){
  For each document:
    if documents contain key
         count++;
    return count;
}
```

Finally, to get tf-idf information, we just multiply them. Algorithm 7 is pseudocode.

Algorithm 7:TF-IDF Algorithm

```
ComputeTFIDF(Map<key, value>, documents){
 Foreach term:
   termTF = ComputeTF(Map<key, value>);
   termIDF = ComputeIDF(Map<key, value>, documents);
   tf-idf = termTF*termIDF;
   retrun tf-idf;
}
```

3.3. Applying Word2Vec

Word2Vec also include two main algorithms, which described in theory part. Word2Vec algorithm described in Algorithm 8.

Algorithm 8:Word2Vec Algorithm

```
Input: data /* the input document*/
windows /*windows size*/
vDim /* dimensionality of the word vector */
nSample /* samples that selected randomly */
Output: updated sync /* word vectors */

repeat until data run out
```

```
// read a sentence from train data
senE] = ReadWords(data);
foreach tWord e sen
  // feed forward
  foreach wWord e sideWords(tWord, windows)
    for d=0 to vDim
      neu1[d] = neu1[d] + syn0[wWo:d][d];
  // negative sampling
  foreach sample e (nSample || tWOrd)
    if sample == tWord then label = true
    else // sample 6 negative samples
      label = false
    for d=0 to vDim do f += neu1[d] *syn1[samp1e][d];
    g = getGradient(f, label);
  for d=0 to vDim do neu1e[d] += 9*syn1[samp1e][d];
  for d=0 to vDim do syn1[samp1e][d] += 9 * neu1[d];
 // backpzopagation
 foreach wWord e sideWords(tWord, windows)
  for d=0 to vDim do syno[wWord][d] += neu1e[d];
```

Example in object oriented code in Example 7.

Example 6: Term-frequency example

```
      Word2Vec vec = new Word2Vec.Builder()
            .minWordFrequency(5)
            .iterations(1)
            .batchSize(250)
            .layerSize(100)
            .lookupTable(table)
            .stopWords(new ArrayList<String>())
            .vocabCache(cache)
            .seed(42)
            .learningRate(0.025)
            .minLearningRate(0.001)
            .sampling(0)
            .windowSize(5)
          .modelUtils(newBasicModelUtils<VocabWord>())
            .iterate(iter)
            .tokenizerFactory(t)
            .build();
    assertEquals(new ArrayList<String>(),vec.getStopWords());
      vec.fit();
    //  WordVectorSerializer.writeWordVectors(vec,
pathToWriteto);
      File tempFile = File.createTempFile("temp", "temp");
      tempFile.deleteOnExit();       WordVectorSerializer.writeFullModel(vec,
```

```
tempFile.getAbsolutePath());
    Collection<String> lst = vec.wordsNearest("day", 10);
```

3. CONCLUSIONS

The work presented in this paper give information about NLP techniques, which can be applied to Russian language based text documents. Given algorithms and examples described for object-oriented programming. This techniques can be used for different goal, but common way is to use this techniques to semantic analysis in machine learning.

ACKNOWLEDGEMENTS

I would like to thank all my teachers from International University of Information Technology for giving a lot of experience and knowledge in master degree courses.

REFERENCES

[1] [1] Scholarpedia, Latent Semantic Analysis, Available: http://www.scholarpedia.org/article/Latent_semantic_analysis [Accessed: 4 March. 2017]

[2] Ms. Anjali Ganesh Jivani (2011), «A Comparative Study of Stemming Algorithms.», IJCTA (2011), pp. 1930-1938

[3] Flavio Chierichetti, Ravi Kumar, Bo Pang (2017), «On the Power Laws of Language: Word Frequency Distributions.», SIGIR'17 (August 7-11, 2017,), pp. 385-394

[4] Jonathan J. Webster, Chunyu Kit (1992), « TOKENIZATION AS THE INITIAL PHASE IN NLP», COLING '92 Proceedings of the 14th conference on Computational linguistics - Volume 4, pp. 1106-1110

[5] Joël Plisson, Nada Lavrac, Dunja Mladenic (2004) « A Rule based Approach to Word Lemmatization», Plisson et al., 2004

[6] Wang Ling, Chris Dyer, Alan Black, Isabel Trancoso (2015) « Two/Too Simple Adaptations of Word2Vec for Syntax Problems», Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pp. 1299–1304