Methods of processing the uzbek language corpus texts

B.B. Elov, Sh.M. Khamroeva, R.H. Alayev, Z.Yu. Khusainova, U.S. Yodgorov

***Abstract*— Computers are designed to process digital or numerical data. However, data is not always in numerical form. How to process data in the form of symbols, words and text? How to teach computers to process our natural language? How do Alexa, Google Home and many other "smart" assistants today understand and respond to our speech? In this article, text processing methods in the field of artificial intelligence, which are called natural language processing, such as Bag-of-words (BOW), CountVectorizer, TF IDF, Co-Occurrence matrix, Word2Vec, CBOW, Skip-Gram, GloVe, ELMO and BERT are presented in order to process the texts of the Uzbek language corpus. The article presents several advantages and disadvantages of the different methods. Methods that generate discrete numerical values of text are easy to understand, implement, and interpret. Algorithms such as TF-IDF can be used to filter simple and non-sense words. Complex tasks in NLP can be solved using distributed text representation algorithms. Distributed text representations can be used to understand and learn a language corpus. These methods are used in the development of modern NLP applications based on CNNs and LSTMs.**

***Keywords*— Uzbek language corpus, text processing, Word2Vec, CBOW, Skip-Gram, GloVe, ELMO, BERT.**

I. INTRODUCTION

Natural language processing is a subfield of artificial intelligence that helps machines understand and process human language. For most natural language processing (NLP) tasks, the most basic step is to convert words into numbers to understand and decode patterns in natural language. In NLP, this step is called **text representation** [1, 2, 3].

The “raw” text in the language corpus is pre-processed and converted into a suitable format for the machine learning model. Data is processed through tokenization, de-wording, punctuation removal, stemming, lemmatization, and a number of other primary processing NLP tasks (Figure 1). In this process, existing "noise" in the data is cleaned [4, 5, 6]. This cleaned data is presented in various forms (templates) according to the input requirements of the NLP application and machine learning model. Common terms used in text processing in NLP are:

Corpus (Corpus, C): a collection of data or multiple textual data together interpreted as a corpus.

Vocabulary (V): collection of all unique words in the corpus.

Document (D): A single text record of a dataset.

Word (Word, W): words in the dictionary.

Figure 1 shows the process of converting the corpus matrix to different input formats for the ML model. Starting from the left, a corpus goes through several steps before obtaining tokens, a set of text building blocks, i.e. words, characters, etc. Since ML models are based on numerical value processing only, the tokens in the sentence are replaced by the corresponding numerical values. In the next step, they are converted to the various input formats shown on the right. Each of these formats has its pros and cons and should be chosen strategically based on the specifics of a given NLP task.

II. LITERATURE OVERVIEW

A. Types of text processing

Although the process of text processing is iterative, it plays an important role for a machine learning model/algorithm. Text views can be divided into two parts [7, 8]:

1. Discrete text representations;
2. Distributed/Continuous text representations.

This article focuses on discrete text representations and introduces text-processing methods using the Python package **Sklearn**.

B. Discrete views of text

In the discrete representation of corpus texts, words in the corpus are represented independently of each other. In this approach, words are represented by indexes corresponding to their position in the vocabulary of the corpus(s). Methods belonging to this category are listed below [1, 3, 7]:

- One-Hot encoding;
- Bag-of-words (BOW);
- CountVectorizer;
- TF-IDF
- Ngram.

Manuscript received September 25, 2023.

Elov Botir Boltayevich – doctor of philosophy (PhD) of technical sciences, associate professor. Tashkent State University of Uzbek Language and Literature named after Alisher Navoi. elov@navoiy-uni.uz

Kamroeva Shahlo Mirdjonovna – doctor of philological sciences (DSc), associate professor. Tashkent State University of Uzbek Language and Literature named after Alisher Navoi. shaxlo.xamrayeva@navoiy-uni.uz

Alayev Ruhillo Habibovich – PhD, National University of Uzbekistan named after Mirzo Ulugbek.
mr.ruhillo@gmail.com

Khusainova Zilola Yuldashevna – PhD student of Tashkent State University of Uzbek Language and Literature named after Alisher Navoi. xusainovazilola@navoiy-uni.uz

Yodgorov Umidjon Saydilla-og li – Teacher of Tashkent State University of Uzbek Language and Literature named after Alisher Navoi. yodgorov@navoiy-uni.uz

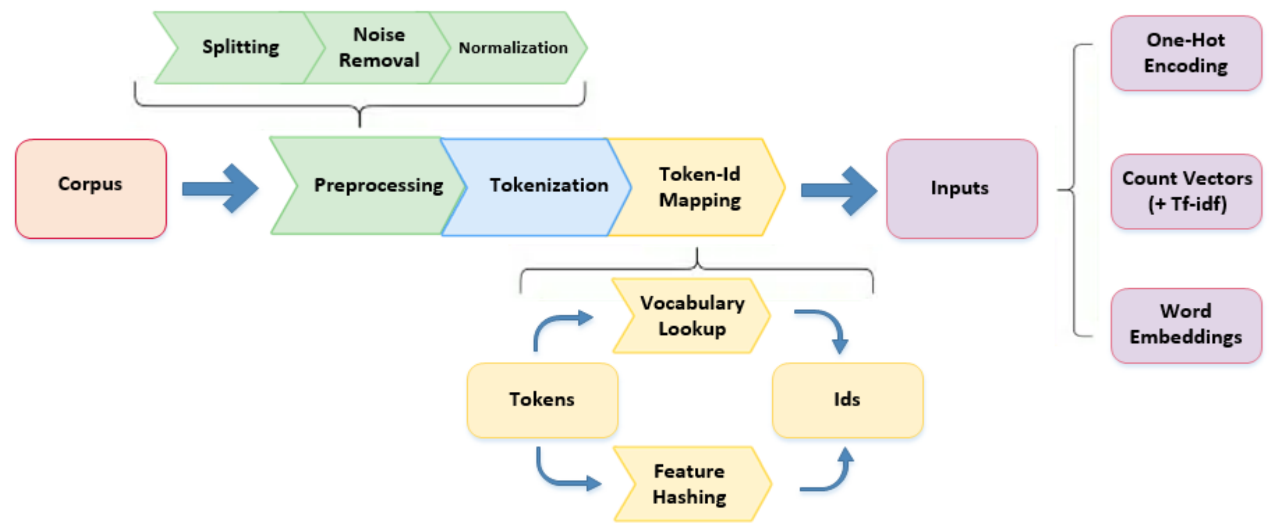

Figure 1. Stages of initial processing of language corpus texts[1]

C. One-Hot encoding method

In the One-Hot encoding method, a vector consisting of 0 and 1 is assigned to each word in the corpus [9]. In the coding of this method, only one element of the vector is assigned - 1, and all other elements - 0. This value represents the element category. The resulting digital vectors are called **hot vectors** in NLP, and a unique **hot vector** is assigned to each word in the corpus. This action allows the machine learning model to recognize each word individually by its vector. One-Hot encoding method can be useful when there is a categorical feature in the data set. For example: The vector values corresponding to the sentence " *Men itimni yaxshi ko'raman* " are expressed as follows for each word in the sentence:

Men → [1 0 0 0], itimni → [0 1 0 0], yaxshi → [0 0 1 0], ko'raman → [0 0 0 1]

Or

$$\begin{array}{r} \textbf{Men:} \\ \textbf{itimni:} \\ \textbf{yaxshi:} \\ \textbf{ko'raman:} \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this case, the sentence is expressed numerically as follows:

sentence = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]

In One-Hot encoding, each bit represents a possible category, and if a given variable does not belong to more than one category, one bit is sufficient to represent it. By this method, the words "Men" and "men" are matched with different vectors. By applying lowercase to all words in word processing, it is possible to match the same vector to uppercase and lowercase letters. In this method, the size of the one-dimensional vector is equal to the size of the dictionary.

When a corpus is encoded using the One-Hot encoding method, each word or token in the dictionary is converted into **a digital vector**. So, sentences in the corpus, in turn, become a matrix of size **(p, q)**. In this,

- "p" is the number of tokens in the sentence;
- "q" is the size of the dictionary.
- The size of the digital vector corresponding to the word in the One-Hot encoding method is directly proportional to the dictionary size of the corpus. So, with the increase in the size of the case, the size of the vector also increases. This method is not useful for large corpora, which may contain up to 100,000 or more unique words. We implement the One-Hot encoding method using the Sklearn package:

```
- from sklearn.preprocessing import
  OneHotEncoder
- import itertools
- # 4 ta namunaviy hujjat
- docs = ['Men NLP bilan ishlayman',
  'NLP juda ajoyib texnologiya',
- 'Tabiiy tilni qayta ishlash',
  'Zamonaviy texnologiyalar bilan
  ishlash']
- # hujjatlarni tokenlarga ajratish
- tokens_docs = [doc.split(" ") for
  doc in docs]
- # tokenlar ro‘yxatini
  umumlashtirish va so'zni
   identifikatoriga moslashtiradigan
  lug'atni yaratish
- all_tokens =
  itertools.chain.from_iterable(toke
  ns_docs)
- word_to_id = {token: idx for idx,
  token in
  enumerate(set(all_tokens))}
- # tokenlar ro‘yxatini token-id
  ro‘yxatlariga aylantirish
```

[1] https://towardsdatascience.com/an-overview-for-text-representations-in-nlp-311253730af1[1]

```
- token_ids = [[word_to_id[token] for
   token in tokens_doc] for
   tokens_doc in tokens_docs]
- # token-id ro'yxatlarini
   umumlashtirish
- vec =
   OneHotEncoder(categories="auto")
- X = vec.fit_transform(token_ids)
- print(X.toarray())
- _________
- [[0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0.
   0. 1. 0.]
-  [0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0.
   0. 0. 1.]
-  [1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1.
   1. 0. 0.]
-  [0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0.
   1. 0. 0.]]
```

Advantages	Disadvantages
Easy to understand and implement	If the number of categories is very large, a large amount of memory is required
	the vector representation of words is orthogonal, and the relationship between different words cannot be determined
	the meaning of the word in the sentence cannot be determined
	a large number of computations are required to represent a high-dimensional sparse matrix

III. Experimental design

A. Bag-of-words method

In the bag-of-words method, words from the corpus are placed in a **"bag of words"** and the frequency of each word is calculated. In this method, word order or lexical information is not taken into account to represent the text. In algorithms based on the BOW method, documents with similar words are returned as similar regardless of word placement.

The BOW method converts a text fragment into **vectors of fixed length**. Word frequency detection helps to compare documents. The BOW method can be used in a variety of NLP applications, such as thematic modeling, document classification, and email spam detection. Below is the BOW vector corresponding to 2 Uzbek sentences.

1-sentence: *"Adirlar ham bahorda lola bilan go'zal, chunki lola – bahorning erka guli".*

2-sentence: *"Lola ham shifokorlik kasbini tanladi".*

	ADIRLAR	BAHORDA	LOLA	GO'ZAL	BAHORNING	ERKA	GULI	SHIFOKORLIK	KASBINI	TANLADI
1-gap	1	1	2	1	1	1	1	0	0	0
2-gap	0	0	1	0	0	0	0	1	1	1

The article "Using bag of words algorithm in natural language processing" written by B.Elov, N.Khudaiberganov and Z.Khusainova presents methods of converting Uzbek texts into digital form using the BoW algorithm [10].

B. Method CountVectorizer

The CountVectorizer method is based on calculating the frequency of word occurrence in the document. Through this method, a matrix of words is defined based on several sentences in the corpus and it is filled with the frequency of each word in the sentence [10]. We implement the CountVectorizer method using the Sklearn package:

```
from sklearn.feature_extraction.text import CountVectorizer
text = ["Men NLP bilan ishlayman. NLP juda ajoyib."]
vectorizer = CountVectorizer()
# Tokenizatsiyalash va lug`atni yaratish
vectorizer.fit(text)
print(vectorizer.vocabulary_)
# hujjatni kodlash
vector = vectorizer.transform(text)
# kodlangan vektorni umumlashtirish
print(vector.shape)
print(vector.toarray())
_________
{'men': 4, 'nlp': 5, 'bilan': 1, 'ishlayman': 2, 'juda': 3, 'ajoyib': 0}
(1, 6)
[[1 1 1 1 1 2]]
```

As it can be clearly seen that the word "NLP" appears twice in the text. In this method, the "weight" of a word in a sentence is equal to its frequency. These weights can be used for different types of analysis and for training ML models. CountVectorizer has parameters like lowercase, strp_accents, preprocessor that can be changed to get the desired results.

Advantages	Disadvantages
Allows to determine the frequency of words in the text	It ignores word location information. The meaning of the word cannot be understood from the result.
The length of the encoded vector is equal to the length of the dictionary	It gives the false conclusion that high-frequency words provide more important information for the text. An example of this is ambiguous words such as "with, and, however,...".
	This method loses the positional information of the word in the sentence.

C. TF-IDF method

In order to identify high frequency words and ignore low frequency words, the "weights" of the words should be normalized accordingly. This task can be performed using the TF-IDF method. The TF-IDF value can be calculated using 2 factors [11,12]:

$$TF-IDF = TF(w,d) * IDF(w)$$

Here, TF(w,d) is the frequency of word "w" in document "d".

The value of IDF(w) can be calculated as:

$$IDF(w) = \log\left(\frac{N}{df(w)}\right)$$

Here, N is the total number of documents and df(w) is the frequency of documents containing the word "w".

The values determined by the TF-IDF method depend on the weight of each word not only on the frequency of words, but also on how often this word occurs in the entire corpus.

To calculate the TF-IDF value, multiply the IDF score by the CountVectorizer value discussed above. From the obtained result, it can be noted that the values for words that occur frequently in the corpus (for example, meaningless words) are relatively large and low for words with very low frequency ("noisy" words). We implement the TF-IDF method using the Sklearn package:

```
from sklearn.feature_extraction.text
import TfidfVectorizer
text1 = ['Men NLP bilan ishlayman', 'NLP
juda ajoyib',
'NLP - bu mashinalarga tabiiy tilni
qayta ishlashga imkon berishdir',
'bu misol nlp texnikasiga namuna']
tf = TfidfVectorizer()
txt_fitted = tf.fit(text1)
txt_transformed =
txt_fitted.transform(text1)
idf = tf.idf_
print(dict(zip(txt_fitted.get_feature_na
mes_out(), idf)))
```

```
{'ajoyib':                 1.916290731874155,
'berishdir': 1.916290731874155, 'bilan':
1.916290731874155,                   'bu':
1.5108256237659907,               'imkon':
1.916290731874155,            'ishlashga':
1.916290731874155,            'ishlayman':
1.916290731874155,                 'juda':
1.916290731874155,          'mashinalarga':
1.916290731874155,                  'men':
1.916290731874155,                'misol':
1.916290731874155,               'namuna':
1.916290731874155,  'nlp':  1.0,  'qayta':
1.916290731874155,               'tabiiy':
1.916290731874155,           'texnikasiga':
1.916290731874155,                'tilni':
1.916290731874155}
```

Lets note the weight of the word "NLP" in the result. Since it is presented in all sentences, it is given a low weight of 1.0. Similarly, the unimportant word "va" is given a relatively low weight of 1.22 because it appears in 3 out of 4 given words.

Similar to the CountVectorizer method, the TF-IDF method has various parameters that can be changed to achieve the desired results. Some important parameters include *lowercase*, *strip_accent*, *stop_words*, *max_df*, *min_df*, *norm*, *ngram_range*, *and sublinear_tf*[2]. The effect of these parameters on the output weight is not considered within the scope of this article.

ADVANTAGES	DISADVANTAGES
Simple, understandable and easy to implement	The positional information of the word is not saved
Common words and low frequency words in the corpus can be identified.	TF-IDF is very dependent on the corps. A high quality educational background is required.
	The semantic features of the words are not recorded.

In the scientific article "Calculating the TF-IDF statistical index for texts of the Uzbek language corpus" written by B. Elov, Z. Husainova and N. Khudaiberganov, the process of sorting documents in the Uzbek language corpus by using the TF-IDF method according to the keyword was considered [11] and 5 stages of TF-IDF value calculation were presented:

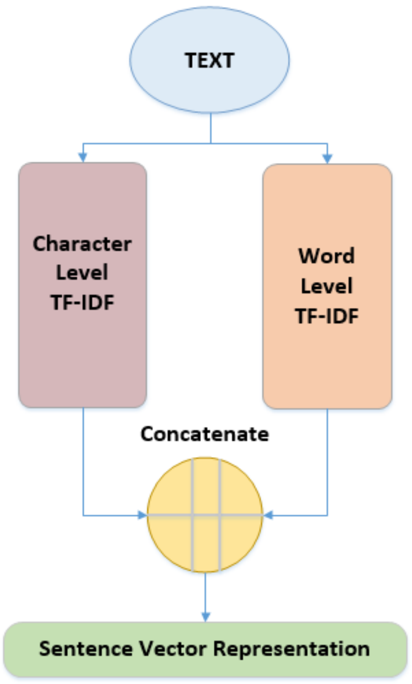

Figure 2. TF-IDF value calculation

A number of scientific conclusions and proposals are given based on the calculations and analyzes carried out by the authors. In particular, it is noted that the use of the TF-IDF method is effective in identifying documents suitable for queries made in large-scale language corpora.

D.Ngram method

The Ngram method is similar to the BoW model, the only difference being that instead of calculating the frequency of a single word, it is the frequency of groups of words that occur together in the corpus (in two or more groups)[13]. Depending on the number of words combined in the text by this method, the model is called bigram (2 words), trigram (3 words).

[2]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
text = ['Men NLP bilan ishlayman', 'NLP
juda ajoyib', 'NLP juda qiyin', 'NLP
keng ommabop']
from sklearn.feature_extraction.text
import CountVectorizer
cv = CountVectorizer(ngram_range=(2,2))
bow = cv.fit_transform(text)
print(cv.vocabulary_)
print(bow[0].toarray())
```

```
{'men nlp': 4, 'nlp bilan': 5, 'bilan
ishlayman': 0, 'nlp juda': 6, 'juda
ajoyib': 1, 'juda qiyin': 2, 'nlp keng':
7, 'keng ommabop': 3}
[[1 0 0 0 1 1 0 0]]
```

ADVANTAGES	DISADVANTAGES
The method of ngrams captures the semantic meaning of a sentence and helps to find the relationship between words.	Out-of-Vocabulary (OOV) words are not processed. If the words do not exist in the dictionary, the relationship between the words or their semantic meanings is not defined.
Easy to implement.	

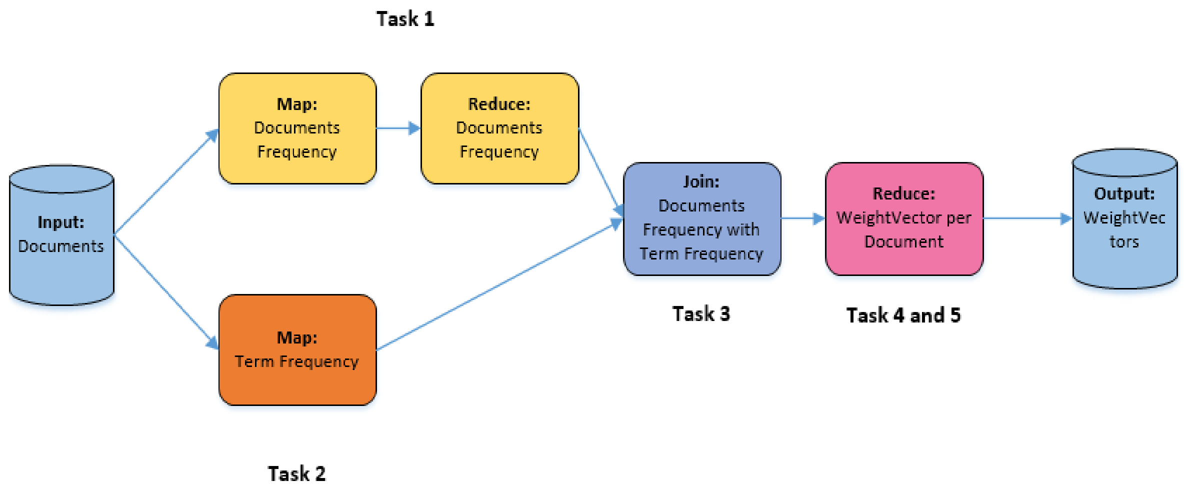

Figure 3. TF-IDF value calculation steps

E. Distributed /continuous text views

A distributed text representation is one in which the numerical representation of a word is independent or non-exclusive of another word, and their configuration often represents different indicators and concepts in the data. In this case, the information about the word is distributed along the corresponding vector. In distributed text representation, each word is different from its discrete representation, which is considered unique and independent of each other.

The most widely used distributed text views today are [14, 15, 16]:

- Co-Occurrence matrix;
- Word2Vec;
- GloVe.

F. Co-Occurrence matrix method

Co-Occurrence matrix method takes into account the co-occurrence of objects located close to each other. An object can be a single word, a bigram (n=2) or a phrase [15,17]. Basically one word is used to calculate matrix values corresponding to a given corpus. This helps us to understand the relationship between different words in the corpus. Let's take the example given in the CountVectorizer method discussed above and convert it to continuous form:

```
from sklearn.feature_extraction.text
import CountVectorizer
import pandas as pd
docs = ['Men NLP bilan ishlayman', 'NLP
juda ajoyib',
'NLP - bu mashinalarga tabiiy tilni
qayta ishlashga imkon berishdir',
'bu misol nlp texnikasiga namuna']
# Nomuhim so'zlarni o'chirish
uz_stop_words=open("uz_stop_words.txt",e
ncoding="utf-8").read().split('\n')
count_vectorizer =
CountVectorizer(stop_words =
uz_stop_words, token_pattern='[a-zA-Z0-
9’‘\-]{1,}')
vectorized_matrix =
count_vectorizer.fit_transform(docs)
co_occurrence_matrix =
(vectorized_matrix.T *
vectorized_matrix)
print(pd.DataFrame(co_occurrence_matrix.
A,

 columns=count_vectorizer.get_feature_na
mes_out(),
 index=count_vectorizer.get_feature_name
s_out()))
```

```
              ajoyib  berishdir  imkon
ishlashga  ishlayman  mashinalarga
ajoyib             1          0      0
0          0            0
berishdir          0          1      1
1          0            1
```

```
imkon                 0          1      1
1          0               1
ishlashga             0          1      1
1          0               1
ishlayman             0          0      0
0          1               0
mashinalarga          0          1      1
1          0               1
misol                 0          0      0
0          0               0
namuna                0          0      0
0          0               0
nlp                   1          1      1
1          1               1
tabiiy                0          1      1
1          0               1
texnikasiga           0          0      0
0          0               0
tilni                 0          1      1
1          0               1
```

The representation of each word is its corresponding row (or column) in the dependency matrix.

Advantages	Disadvantages
Expresses the connection of words more simply	A matrix is generated similar to the CountVectorizer and TF-IDF matrices
Unlike discrete methods, preserves the order of words in a sentence	The size of the matrix depends on the size of the dictionary
Determines the interconnection of words from the whole corpus	It is impossible to identify all word combinations by using this method.

G.Word2Vec method

Word2Vec is a popular word embedding algorithm. This algorithm was developed by Thomas Mikalov in 2013 under the research "Efficient evaluation of word representation in vector space" [18,19]. The method is based on prediction of word expression.

Word embedding is a vector representation of a word, which is represented by a defined vector dimension, taking into account the semantic and syntactic relationship of each word with other words. Word2vec architecture is a single hidden layer network. The weight of the hidden layer is determined by *the word loss function (normal backprop).*

This architecture is similar to an autoencoder, where you have an encoder layer and a decoder layer, and the middle part is a compressed representation of the input that can be used for dimensionality reduction or anomaly detection. Corpus representation using the Word2vec method is performed in 2 different ways [20, 21]:

– CBOW is based on predicting an intermediate word based on the surrounding word context. The CBOW method attempts to fill in the blanks based on which word is most appropriate in the context (taking into account the surrounding words). This method provides efficient results with smaller data sets.

– Skip-Gram – tries to guess the surrounding context words from the target word (opposite of CBOW). Performs better on larger datasets. However, it takes a lot of time to process the training data.

The degree of similarity between words is determined using vector arithmetic through the Word2vec method. From a template like **"Man is to woman as king is to queen"**, it is possible to get a result like **"king" = "man" + "woman" = "queen"** through arithmetic operations. Also, the word "queen" in this sentence represents syntactic and semantic relations. Let's look at the word2vec method using the gensim package:

```
from gensim.models import Word2Vec
sentences = ['Men NLP bilan ishlayman',
'NLP juda ajoyib',
'NLP - bu mashinalarga tabiiy tilni
qayta ishlashga imkon berishdir',
'bu misol nlp texnikasiga namuna']
# gapni oldindan qayta ishlash Word2Vec
uchun zarur formatga aylantirish
sentence_list=[]
for i in sentences:
    li = list(i.split(" "))
    sentence_list.append(li)
model = Word2Vec(sentence_list,
min_count=1,
                workers=4, sg=1,
window=4)
model.wv['nlp']
model.wv.most_similar(positive=['nlp'])
```

```
[('imkon', 0.24666069447994232),
 ('Men', 0.11936754733324051),
 ('ajoyib', 0.11928389966487885),
 ('ishlashga', 0.11663015931844711),
 ('texnikasiga', 0.09614861011505127),
 ('bu', 0.08543577790260315),
 ('ishlayman', 0.07172605395317078),
 ('tilni', 0.05970853567123413),
 ('mashinalarga', 0.04119439423084259),
 ('-', 0.012471411377191544)]
```

In just a few lines of the above program code, we are able to not only train and display words as a vector, but also identify similar and different words. There are two ways to determine the similarity between vectors:

– **Normalized:** by calculating the scalar product between the vectors, it is possible to determine their similarity;

– **Unnormalized**: the cosine similarity between vectors can be calculated using the following formula:

$$\mathit{cosine\ similarity} = 1 - \mathit{cosine\ distance} = \frac{u * v}{||u|| * ||v||}$$

Algorithms for determining digital vectors based on the corpus, machine learning of the corpus, and determining relationships between words will be discussed in later scientific publications. The advantages and disadvantages of the Word2Vec method are listed in the following table:

Advantages	Disadvantages
It allows to determine the syntactic and semantic relations between different words	Out-of-vocabulary words (OOV) cannot be recycled.
The size of the digital vector corresponding to the word is small and flexible.	The semantic representation of a word is based only on its neighbors.
Corpus training process does not depend on the human factor.	To apply the Word2Vec method to a new natural language, it is necessary to perform many steps.

A larger corpus is required to improve data accuracy.

H. The GloVe method

Global Vectors, or GloVe for short, is a modern NLP method of numerically representing words. This method was developed and implemented by Jeffery Pennington, Richard Socher, and Christopher Manning in 2014 [22]. Unlike the word2vec method mentioned above, this method studies the local and global statistics of a word and is called a hybrid approach to word representation. The GloVe method uses the following notations:

$$v_i^T v_j = logP(i|j)$$

Or

$$v_i^T v_j = logP\left(X_{ij}\right) - logP\left(X_i\right)$$

Thus, corresponding to P(i|j), Vi and Vj are the values of the word vectors. Together, these vectors represent the global statistics in the colocation matrix. Information about the objective function in the GloVe method will be provided in the next scientific articles. The formation and use of GloVe vectors based on pre-trained models of large volumes of text is given below:

```
import gensim.downloader as api
# 2 milliard tvitning 25 o'lchamli GloVe tasvirini yuklab olish
twitter_glove = api.load("glove-twitter-25")
# O'xshash so'zlarni topish
print(twitter_glove.most_similar("book", topn=10))
# 25D vektorlarni olish
print(twitter_glove['book'])
print(twitter_glove.similarity("book", "school"))
```

```
[('books', 0.94181889295578),
('project', 0.9214614033699036),
('review', 0.9140495657920837),
('script', 0.9069417119026184), ('new',
0.9069172143936157), ('feature',
0.8995184302330017), ('guest',
0.897861659526825), ('read',
0.8931056261062622), ('post',
0.8916701674461365), ('art',
0.8880472183227539)]
[ 0.21621   0.056781  0.82955  -0.1424
0.82832  -0.87341   1.699
 -0.25702   0.65303  -0.82435   0.26496
0.4612   -4.0463   -0.044556
  0.15648  -0.083655  0.72399   0.20802
-0.27561  -0.024987 -0.83992
 -0.92536  -0.95454   0.42348  -0.14709
]
0.7545484
```

The advantages and disadvantages of the GloVe method are listed in the table below:

ADVANTAGES	DISADVANTAGES
It performs better than the Word2vec method	Due to the use of co-occurrence matrix and global information, the GloVe method requires much more memory than the word2vec method.
Considers word pairs and word pair relationships when constructing vectors	Similar to the word2vec method, it does not solve the ambiguous word problem
Compared to Word2Vec, the GloVe method is easier to parallelize, so the training time is shorter	

IV. DISCUSSION

A. Modern approaches

1) ELMO method

In March 2018, Matthew Peters et al presented a paper entitled Deep Contextual Word Representations [23]. His proposed method tries to overcome the shortcomings of the word2vec and GloVe methods by having a many-to-one relationship between the vector representation and the word it represents. In the ELMO method, the vector representation of the word is modified accordingly, taking into account the context.

The ELMO method uses character-level CNNs to transform words into initial word vectors. Additionally, two-way LSTMs are used in the training process. In the method, a combination of forward and backward iteration creates intermediate word vectors representing pre- and post-word context information, respectively. The weighted sum of the initial word vector and the 2 intermediate word vectors gives the final value.

2) BERT method

BERT is a method for pre-training deep bidirectional transformers for language understanding, described in the 2019 Google AI team paper "Pre-training of Deep Bidirectional Transformers for Language Understanding" [24]. This is a new self-supervised machine-learning task for pre-training transformers.

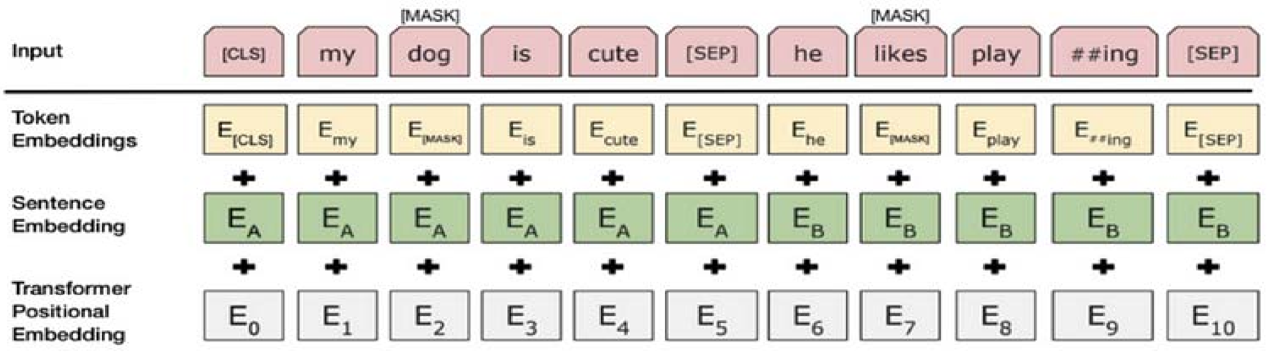
Input
[CLS]
my
[MASK]
dog
is
cute
[SEP]
he
[MASK]
likes
play
##ing
[SEP]
Token Embeddings
E[CLS] Emy E[MASK] Eis Ecute E[SEP] Ehe E[MASK] Eplay E##ing E[SEP]
Sentence Embedding
EA EA EA EA EA EA EB EB EB EB EB
Transformer Positional Embedding
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10

Figure 4. BERT method architecture.

The BERT method uses the dual context of the language model. It attempts left-to-right and right-to-left "masking" to generate intermediate tokens used for prediction tasks.

The input to the BERT model consists of token placement, segmentation, and follows a masking strategy for the model to correctly predict the word in context. It uses a matched transformer network to perform other tasks such as NER and question-and-answer systems, which learns the contextual relationship between words through the BERT method.

3) Digital text display applications/ Applications of digital display of text

The numerical models of text presented in this article can be applied to the following NLP tasks:

– Text Classification: In the task of text classification, it is important to form the text in vector form for the initial processing of the text.
– Topic Modelling: Topic Modeling requires that the text needs to be presented in the correct format for modeling different topics.
– Autocorrect Model: Spelling errors in the text are corrected through the autocorrect model. The text provided by the autocorrect model tool must be presented in the required numeric format.
– New text generation (Text Generation): Probabilistic numerical text format is required for text generation.

Before training a machine learning model, it is important to represent the text in a specific format. The more complex the format, the better the accuracy of the model and the better the results. Every NLP application that involves textual data requires a good text representation.

V. Conclusion

Through discrete text representation methods, each word in the corpus is considered unique and converted into a numerical form based on the various methods discussed above. The article presents several advantages and disadvantages of the different methods. We summarize them as a whole. Methods that generate discrete numerical values of text are easy to understand, implement, and interpret. Algorithms such as TF-IDF can be used to filter simple and non-sense words. And it, in turn, helps to train and generalize the model faster. The direct proportionality of the vocabulary to the size of the corpus can be cited as a disadvantage of the methods. A large dictionary can cause various memory limitations. In all methods, the words in the corpus are considered to be independent from each other. This leads to the generation of very sparse vectors with non-zero values. The generated vectors do not represent the context or semantics of the word. Discrete representations of text are widely used in classical machine learning methods and deep learning applications to solve NLP tasks such as document similarity, sentiment classification, spam classification, and topic modeling.

Complex tasks in NLP can be solved using distributed text representation algorithms. Distributed text representations can be used to understand and learn a language corpus. An example of this is the study of words within a corpus and how they relate to each other. Today, distributed text representations are widely used in the development of supervised learning models to solve complex NLP tasks such as Q&A systems, document classification, chatbot, NER object recognition. Currently, these methods are used in the development of modern NLP applications based on CNNs and LSTMs.

References

[1] Naseem, U., Razzak, I., Khan, S. K., & Prasad, M. (2021). A Comprehensive Survey on Word Representation Models: From Classical to State-of-the-Art Word Representation Language Models. *ACM Transactions on Asian and Low-Resource Language Information Processing, 20*(5). https://doi.org/10.1145/3434237

[2] Chai, C. P. (2023). Comparison of text preprocessing methods. *Natural Language Engineering, 29*(3). https://doi.org/10.1017/S1351324922000213

[3] Probierz, B., Hrabia, A., & Kozak, J. (2023). A New Method for Graph-Based Representation of Text in Natural Language Processing. *Electronics, 12*(13). https://doi.org/10.3390/electronics12132846

[4] B.ELov, E.Adalı, Sh.Khamroeva, O.Abdullayeva, Z.Xusainova, N.Xudayberganov (2023). The Problem of Pos Tagging and Stemming for Agglutinative Languages. *8 th International Conference on Computer Science and Engineering UBMK 2023, Mehmet Akif Ersoy University, Burdur – Turkey.*

[5] B.ELov, Sh.Khamroeva, Z.Xusainova (2023). The pipeline processing of NLP. *E3S Web of Conferences 413, 03011, INTERAGROMASH 2023.* https://doi.org/10.1051/e3sconf/202341303011

[6] B.Elov, Sh.Hamroyeva, X.Axmedova. Methods for creating a morphological analyzer. *14th International Conference on Intellegent Human Computer Interaction, IHCI 2022, 19-23 October 2022, Tashkent.* https://dx.doi.org/10.1007/978-3-031-27199-1_4

[7] Siebers, P., Janiesch, C., & Zschech, P. (2022). A Survey of Text Representation Methods and Their Genealogy. *IEEE Access, 10*. https://doi.org/10.1109/ACCESS.2022.3205719

[8] Jiang, Z., Gao, S., & Chen, L. (2020). Study on text representation method based on deep learning and topic information. *Computing, 102*(3). https://doi.org/10.1007/s00607-019-00755-y

[9] Rodríguez, P., Bautista, M. A., Gonzàlez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing, 75*. https://doi.org/10.1016/j.imavis.2018.04.004

[10] B.Elov, Z.Xusainova, N.Xudayberganov. Tabiiy tilni qayta ishlashda Bag of Words algoritmidan foydalanish. *O`zbekiston: til va madaniyat (Amaliy filologiya), 2022, 5(4).* http://aphil.tsuull.uz/index.php/language-and-culture/article/download/32/29

[11] B.Elov, Z.Xusainova, N.Xudayberganov. O`zbek tili korpusi matnlari uchun TF-IDF statistik ko`rsatkichni hisoblash. *SCIENCE AND INNOVATION INTERNATIONAL SCIENTIFIC JOURNAL VOLUME 1 ISSUE 8 UIF-2022: 8.2 | ISSN: 2181-3337*

[12] *https://www.academia.edu/105829396/OZBEK_TILI_KORPUSI_MATNLARI_UCHUN_TF_IDF_STATISTIK_KORSATKICHNI_HISOBLASH*

[13] Fu, Y., & Yu, Y. (2020). Research on text representation method based on improved TF-IDF. *Journal of Physics: Conference Series, 1486*(7). https://doi.org/10.1088/1742-6596/1486/7/072032

[14] Maharjan, S., Mave, D., Shrestha, P., Montes-Y-Gómez, M., González, F. A., & Solorio, T. (2019). Jointly learning author and annotated character N-gram embeddings: A case study in literary text. *International Conference Recent Advances in Natural Language Processing, RANLP*, *2019-September*. https://doi.org/10.26615/978-954-452-056-4_080

[15] Wawrzyński, A., & Szymański, J. (2021). Study of statistical text representation methods for performance improvement of a hierarchical attention network. *Applied Sciences (Switzerland), 11*(13). https://doi.org/10.3390/app11136113

[16] Zhao, J. S., Song, M. X., Gao, X., & Zhu, Q. M. (2022). Research on Text Representation in Natural Language Processing. *Ruan Jian Xue Bao/Journal of Software*, *33*(1). https://doi.org/10.13328/j.cnki.jos.006304

[17] Babić, K., Martinčić-Ipšić, S., & Meštrović, A. (2020). Survey of neural text representation models. In *Information (Switzerland)* (Vol. 11, Issue 11). https://doi.org/10.3390/info11110511

[18] Eleyan, A., & Demirel, H. (2011). Co-occurrence matrix and its statistical features as a new approach for face recognition. *Turkish Journal of Electrical Engineering and Computer Sciences*, *19*(1). https://doi.org/10.3906/elk-0906-27

[19] Cahyani, D. E., & Patasik, I. (2021). Performance comparison of tf-idf and word2vec models for emotion text classification. *Bulletin of Electrical Engineering and Informatics*, *10*(5). https://doi.org/10.11591/eei.v10i5.3157

[20] Method, N. W., Goldberg, Y., Levy, O., Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2014). word2vec Explained : Deriving Mikolov et al. *ArXiv:1402.3722 [Cs, Stat]*, *2*.

[21] Xiong, Z., Shen, Q., Xiong, Y., Wang, Y., & Li, W. (2019). New generation model of word vector representation based on CBOW or skip-gram. *Computers, Materials and Continua*, *60*(1). https://doi.org/10.32604/cmc.2019.05155

[22] Jang, B., Kim, I., & Kim, J. W. (2019). Word2vec convolutional neural networks for classification of news articles and tweets. *PLoS ONE*, *14*(8). https://doi.org/10.1371/journal.pone.0220976

[23] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. https://doi.org/10.3115/v1/d14-1162

[24] Kutuzov, A., & Kuzmenko, E. (2021). Representing ELMo embeddings as two-dimensional text online. *EACL 2021 - 16th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the System Demonstrations*. https://doi.org/10.18653/v1/2021.eacl-demos.18

[25] Joshi, M., Levy, O., Weld, D. S., & Zettlemoyer, L. (2019). BERT for coreference resolution: Baselines and analysis. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*. https://doi.org/10.18653/v1/d19-1588