PROCESSING THE UZBEK LANGUAGE CORPUS TEXTS

Anvar Shukhratbekovich Abdullaev
Urgench Branch of Tashkent University of Information Technologies
anvar.abdullayev.1989@gmail.com

ABSTRACT

This paper investigates the application of various text processing techniques within the realm of artificial intelligence, with a particular focus on natural language processing (NLP) for the Uzbek language. It discusses methods such as Bag-of-Words, CountVectorizer, TF-IDF, Co-Occurrence Matrix, Word2Vec, CBOW, Skip-Gram, GloVe, ELMO, and BERT, and evaluates their advantages and disadvantages in the context of text representation. The paper highlights the importance of discrete numerical representations of text for simplicity, ease of implementation, and interpretability, and emphasizes the significance of distributed text representation algorithms for complex NLP tasks.

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

INTRODUCTION

Natural language processing (NLP) is an essential subfield of artificial intelligence that enables machines to understand and analyze human language. Central to NLP is the task of transforming words into numerical representations to identify patterns in natural language. This process, known as text representation, involves preprocessing raw text from language corpora and converting it into machine-readable formats. The preprocessing steps include tokenization, de-wording, punctuation elimination, stemming, lemmatization, and more, which help remove noise from the data. The cleaned data is then structured into various formats suitable for NLP applications and machine learning models. This paper discusses common terms in NLP text processing, such as corpus, vocabulary, document, and word, and illustrates the process of converting a corpus matrix into different input formats for machine learning models.

LITERATURE REVIEW

A. Text Processing Categories

Text processing, an iterative endeavor, holds significant importance for machine learning models/algorithms. Textual representations can be broadly categorized into two types:

1. Discrete Text Representations
2. Distributed/Continuous Text Representations

This paper primarily concentrates on discrete text representations, presenting text-processing techniques utilizing the Python library **Sklearn**.

B. Discrete Text Representations

In the discrete representation of corpus texts, words within the corpus are depicted independently of each other. Under this paradigm, words are encoded using indexes corresponding to their positions within the corpus vocabulary. Methods falling under this category include

- One-Hot Encoding
- Bag-of-Words (BOW)
- CountVectorizer
- TF-IDF
- Ngram

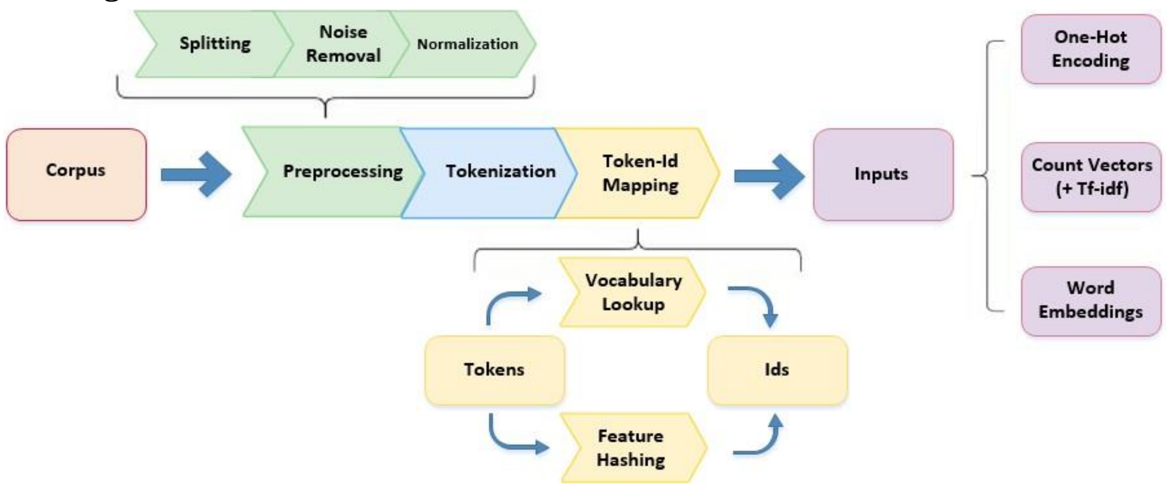


Fig. 1. Stages of initial processing of language corpus texts

C. One-Hot Encoding Method

One-Hot encoding assigns a binary vector comprising 0s and 1s to each word in the corpus. In this encoding scheme, only one element in the vector is assigned a value of 1, indicating the category of the element. The resultant binary vectors, termed "hot vectors" in NLP, assign a unique hot vector to each word in the corpus. This facilitates individual word recognition by the machine learning model based on its vector representation. . For example: The vector values corresponding to the sentence " Men matematikani yaxshi ko'raman " are expressed as follows for each word in the sentence:

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

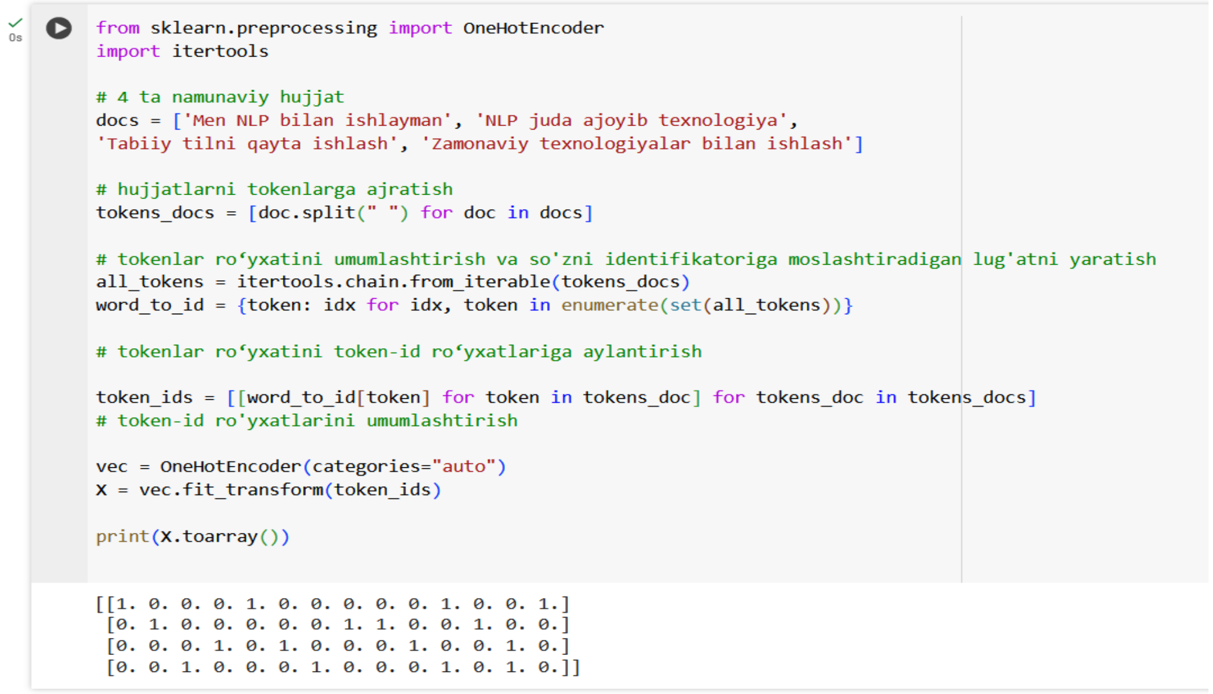
sentence = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]

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.

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 roʻyxatini umumlashtirish va so'zni identifikatoriga moslashtiradigan lug'atni yaratish
all_tokens = itertools.chain.from_iterable(tokens_docs)
word_to_id = {token: idx for idx, token in enumerate(set(all_tokens))}

# tokenlar roʻyxatini token-id roʻyxatlariga aylantirish

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())
```

```
[[1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0.]]
```

Fig. 2. Implementing the One-Hot encoding method using the Sklearn package

ADVANTAGES:

- Simple to get it and execute

DISADVANTAGES:

- On the off chance that the number of categories is exceptionally huge, an expansive sum of memory is required;
- The vector representation of words is orthogonal, and the relationship between distinctive words cannot be decided;
- The meaning of the word within the sentence cannot be decided; an expansive number of computations are required to speak to a high-dimensional inadequate framework

EXPERIMENTAL DESIGN

A. Bag-of-words method

In the bag-of-words (BOW) technique, words extracted from the corpus are pooled into a "bag of words," and the frequency of each word is tallied. This method disregards word order or lexical nuances in text representation. Algorithms utilizing the BOW approach consider documents containing similar words as equivalent, regardless of the arrangement of these words.

The BOW method translates a text segment into fixed-length vectors, facilitating comparison between documents. It finds utility across various NLP applications, including 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”.*

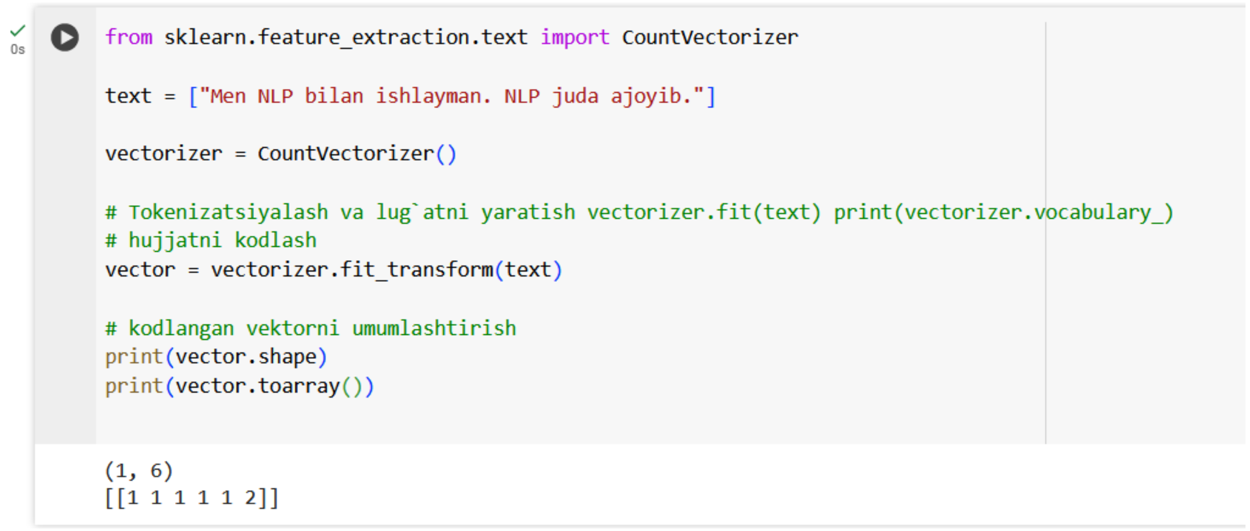
2-sentence: *“Lola ham shifokorlik kasbini tanladi”.*

	ADIRLAR	BAHORDA	LOLA	GO‘ZAL	BAHORNING	ERKA	GULI	SHIFOKORLI	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

B. CountVectorizer Method

The CountVectorizer technique operates by computing the frequency of word occurrences within a document. This method involves constructing a matrix of words derived from multiple sentences within the corpus, with each cell populated by the frequency of the respective word in the sentence.

It is implemented using the Sklearn package in Python. This approach emphasizes the significance or "weight" attributed to a word within a sentence, corresponding to its frequency. These weights are instrumental in training machine learning models and serve various analytical purposes. CountVectorizer offers configurable parameters such as lowercase, strip_accents, and preprocessor, which can be adjusted to achieve specific desired outcomes.



```
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.fit_transform(text)

# kodlangan vektorni umumlashtirish
print(vector.shape)
print(vector.toarray())
```

```
(1, 6)
[[1 1 1 1 1 2]]
```

Fig. 3. Implementing the CountVectorizer method using the Sklearn package

Advantages:

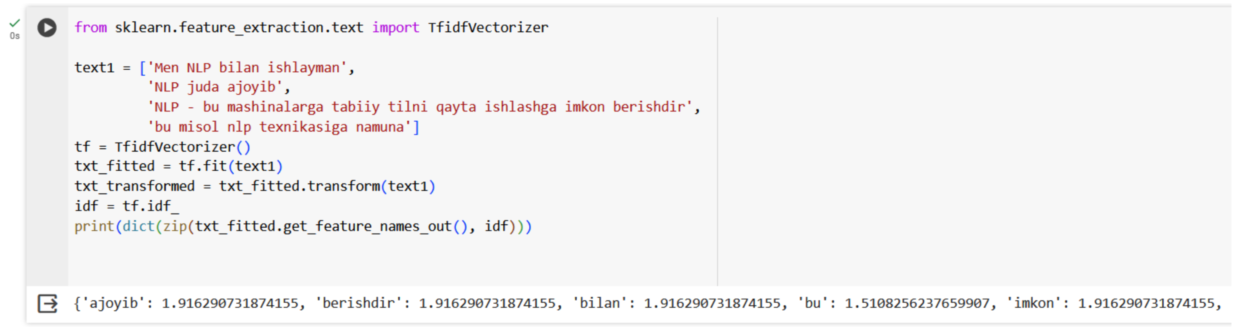
- Facilitates the determination of word frequencies within the text.
- The length of the encoded vector matches the size of the dictionary.

Disadvantages:

- Disregards word location details, hindering the interpretation of word meanings.
- Erroneously implies that high-frequency words offer more significant information for the text, which may not always hold true, especially with ambiguous words like "with," "and," "however," etc.
- Loss of positional information regarding the word's placement within the sentence.

C. TF-IDF Method

To discern high-frequency words while disregarding low-frequency ones, it's essential to normalize the "weights" of words accordingly. This normalization task is effectively accomplished through the employment of the TF-IDF method. The TF-IDF value can be calculated using two factors: TF(wd), which is the frequency of word "w" in document "d," and IDF(w), which is calculated based on the total number of documents and the frequency of documents containing the word "w." $IDF(w) = \log(\frac{N}{df(w)})$. The TF-IDF method assigns values based not only on word frequency but also on their occurrence across the entire corpus. To compute the TF-IDF value, the IDF score is multiplied by the CountVectorizer value, as previously mentioned. The resulting values indicate relatively high scores for words occurring frequently in the corpus, such as meaningless words, while words with minimal frequency, often termed as "noisy" words, receive lower scores. $TF - IDF = TF(w,d) * IDF(w)$ We implement the TF-IDF method using the Sklearn package in Python.



```
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_names_out(), idf)))
```

```
{'ajoyib': 1.916290731874155, 'berishdir': 1.916290731874155, 'bilan': 1.916290731874155, 'bu': 1.5108256237659907, 'imkon': 1.916290731874155,
```

Fig. 4. Implementing the TF-IDF method using the Sklearn package

{'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}

Let's note the weight of the word "NLP" within the result. Since it is displayed in all sentences, it is given a moo weight of 1.0. Essentially, the insignificant word "va" is given a moderately moo weight of 1.22 since it shows up in 3 out of 4 given words.

Comparable to the CountVectorizer strategy, the TF-IDF strategy has different parameters that can be changed to realize the required comes about. A few imperative parameters incorporate lowercase, strip_accent, stop_words, max_df, min_df, norm, ngram_range, and sublinear_tf. The impact of these parameters on the yield weight isn't considered inside the scope of this article.

ADVANTAGES: Straightforward, justifiable and simple to execute; Common words and moo frequency words within the corpus can be distinguished.

DISADVANTAGES: The positional data of the word isn't spared; TF-IDF is exceptionally subordinate on the corps. A high-quality instructive foundation is required; The semantic highlights of the words are not recorded. Within the logical article "Calculating the TF-IDF factual list for writings of the Uzbek dialect corpus" composed by B. Elov, Z. Husainova and N. Khudaiberganov, the method of sorting archives within the Uzbek dialect corpus by utilizing the TF-IDF strategy concurring to the keyword was considered and 5 stages of TF-IDF esteem calculation were displayed:

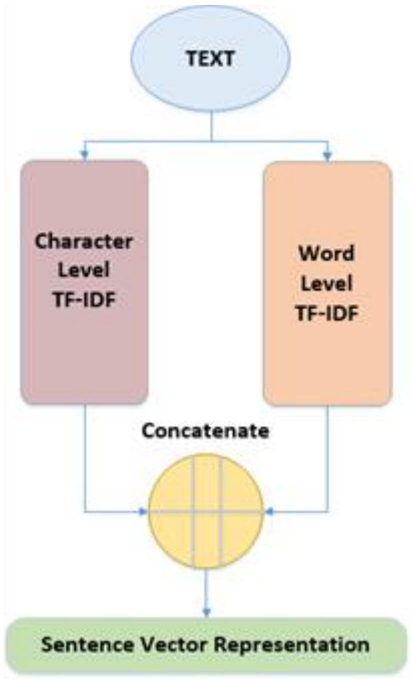
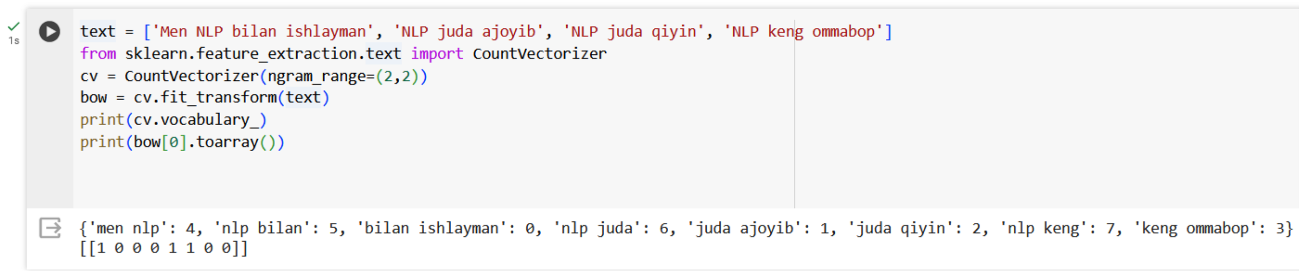


Fig. 5. TF-IDF value calculation

A number of logical conclusions and recommendations are given based on the calculations and analyzes carried out by the creators. In specific, it is famous that the utilize of the TF- IDF strategy is viable in recognizing reports *reasonable for questions made in large-scale dialect corpora.*

D.Ngram method

The Ngram approach resembles the Bag of Words (BoW) model, with the key distinction lying in the calculation of frequencies. Instead of tallying the occurrence of individual words, it computes the frequency of word combinations appearing together in the corpus, known as bigrams (two words) or trigrams (three words), depending on the number of words grouped together in the text.



```
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]]
```

Fig. 6. Using Bag of Words method with CountVectorizer

E. Distributed /continuous text views

A distributed text representation refers to a numerical portrayal of words that isn't tied or restricted to one another, and their arrangement typically reflects various signals and ideas within the data. Here, details about each word are spread across its respective vector. Unlike discrete representation where each word is considered distinct and unrelated to others, distributed text representation ensures each word is unique in its own right. The prevailing distributed text representations in current usage encompass:

- Co-Occurrence matrix
- Word2Vec
- *GloVe*

F. Co-Occurrence matrix method

The Co-Occurrence matrix technique considers the proximity of objects occurring together. These objects could be individual words, bigrams (n=2), or phrases. Essentially, each word is utilized to compute matrix values for a given corpus, aiding in comprehending the associations among various words within the corpus. To illustrate, let's transform the example provided in the previously mentioned CountVectorizer method into a continuous form.

ADVANTAGES: The ngrams approach grasps the semantic essence of a sentence and facilitates the identification of connections among words.

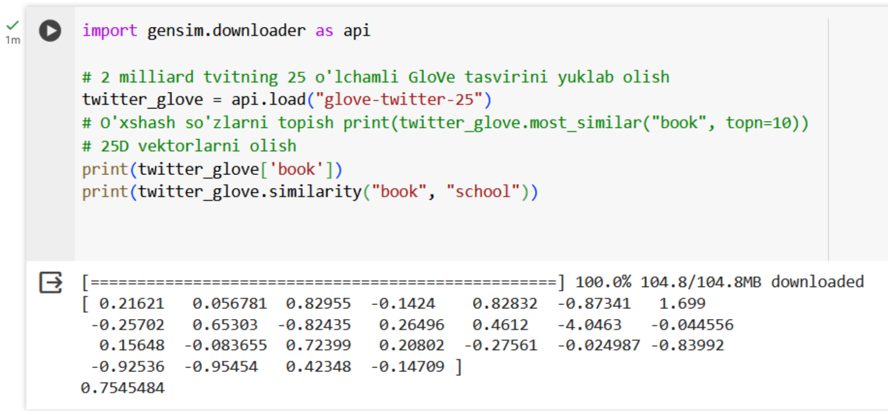
DISADVANTAGES: Words that are Out-of-Vocabulary (OOV) are not handled or analyzed; If the words are absent from the dictionary, the connection or semantic significance between them is not established.

G. Word2Vec

Word2Vec is a popular technique that utilizes neural networks to learn word embeddings. It consists of two main models: Continuous Bag of Words (CBOW) and Skip-Gram. Both models aim to predict words based on their context, but they differ in their approach. CBOW predicts a target word based on its surrounding context words, while Skip-Gram predicts the context words given a target word.

H.GloVe

Global Vectors for Word Representation (GloVe) is another method for obtaining distributed word representations. It combines the advantages of matrix factorization and context window methods by using a co-occurrence matrix and factorizing it to obtain dense word vectors. GloVe captures both global statistics and local context information of words in the corpus. $v_i^T v_j = \log P(i|j)$ or $v_i^T v_j = \log P(X_{ij}) - \log P(X_i)$



```
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"))
```

```
[==================================================] 100.0% 104.8/104.8MB downloaded
[ 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
```

Fig. 7. Using of GloVe vectors based on pre-trained models of large volumes of text

Therefore, denoting P(i|j), Vi and Vj represent the values of the word vectors. These vectors collectively embody the overall statistics within the co-location matrix. Details regarding the objective function in the GloVe approach will be elucidated in forthcoming scientific publications. The creation and application of GloVe vectors using pre-trained models on extensive text corpora are outlined.

DISCUSSION

A. Modern approaches

1) ELMO

Embeddings from Language Models (ELMO) is a deep contextualized word representation approach. It utilizes a bidirectional LSTM (Long Short-Term Memory) model to generate word embeddings that consider the entire context of a word within a sentence. ELMO embeddings are dynamic and change based on the surrounding words, providing a rich representation of word meanings.

2) BERT

Bidirectional Encoder Representations from Transformers (BERT) is a powerful technique for generating contextualized word embeddings. BERT employs the Transformer architecture to process words in parallel and capture the context of a word from both the left and right sides of a sentence. BERT has achieved state-of-the-art results in various NLP tasks and is widely used in the industry.

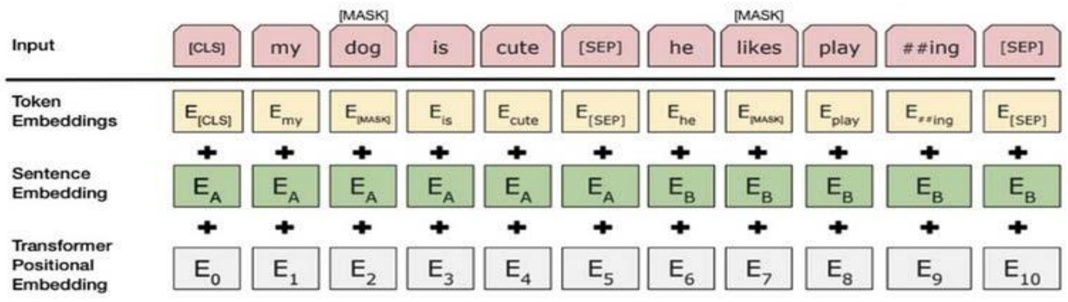


Fig. 4. BERT method architecture.

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

The numerical text models discussed in this article are applicable to various NLP tasks, including:

- Text Classification: Text classification tasks necessitate converting text into vector form for initial processing.
- Topic Modelling: Effective topic modeling relies on presenting text in the appropriate format for modeling diverse topics.
- Autocorrect Model: Correcting spelling errors in text involves employing an autocorrect model, which requires text to be formatted in the specified numeric format.
- New Text Generation (Text Generation): Probabilistic numerical text formatting is essential for generating new text.

Before training a machine learning model, it's crucial to represent text in a specific format. The complexity of the format directly impacts the accuracy of the model and the quality of results. Effective text representation is fundamental to every NLP application involving textual data.

Each of these distributed text representation techniques offers unique advantages in capturing semantic relationships between words and improving the performance of NLP models. The choice of technique depends on the specific requirements of the NLP task and the characteristics of the dataset.

CONCLUSION

Through discrete text representation methods, each word within the corpus is treated as unique and transformed into a numerical format using various techniques outlined earlier. The article outlines several advantages and disadvantages of these methods, which we summarize collectively. Methods that produce discrete numerical representations of text are straightforward to comprehend, implement, and interpret. Techniques like TF-IDF aid in filtering out trivial and nonsensical words, thereby accelerating model training and generalization. However, a drawback of these methods lies in the direct correlation between vocabulary size and corpus size, potentially leading to memory constraints with larger dictionaries. Moreover, in all these methods, words in the corpus are assumed to be independent, resulting in sparse vectors with non-zero values that fail to capture word context or semantics.

Discrete text representations find extensive application in classical machine learning techniques and deep learning approaches for addressing NLP tasks such as document similarity assessment, sentiment analysis, spam detection, and topic modeling.

For more complex NLP tasks, distributed text representation algorithms offer solutions. These algorithms enable a deeper understanding of language corpora, such as exploring word relationships within a corpus. Distributed text representations are widely adopted in the development of supervised learning models to tackle intricate NLP challenges like question answering systems, document categorization, chatbots, and named entity recognition (NER). Presently, these methods are integrated into modern NLP applications leveraging Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs).

REFERENCES

1. 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.
2. 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
3. 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
4. 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
5. 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

6. 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
7. A. S. Abdullayev, O. T. Allamov, O. A. Rustamov, R. J. Urinov and Y. U. Kadamovna, "Analysis of existing smart crossroads and a new approach of smart crossroad," 2021 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2021, pp. 1-5, doi: 10.1109/ICISCT52966.2021.9670167.
8. A. K. Nishanov, O. T. Allamov, O. B. Ruzibaev, A. S. Abdullaev and S. T. Allamova, "An approach to finding the most optimal route in a dynamic graph," 2021 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2021, pp. 01-05, doi: 10.1109/ICISCT52966.2021.9670385.
9. O. K. Khujaev, O. Rustamov and A. Abdullayev, "The use of artificial intelligence in the implementation of the multilingual website of the dorul hikmat project," 2022 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2022, pp. 1-2, doi: 10.1109/ICISCT55600.2022.10146980.
10. Chai, C. P. (2023). Comparison of text preprocessing methods. Natural Language Engineering,29(3). https://doi.org/10.1017/S1351324922000213
11. Probierz, B., Hrabia, A., & Kozak, J. (2023). A New Method for Graph-Based Representation of Text in Natural Language Processing.
a. Electronics, 12(13). https://doi.org/10.3390/electronics12132846
12. 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
13. 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
14. 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