JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

A Survey On Text Preprocessing, Data Vectorization And Recent NLP Models

[1]Ayush Shah, [2]Ayushi S, [3]Akshata G, [4]Dr.Preeti Satish

[1]Student, [2]Student, [3]Professor, [4]Professor
[1]Department of Computer Science and Engineering,
[1]Dayanada Sagar College of Engineering, Bangalore, India

Abstract: This paper is a survey on different NLP (Natural Language processing) methods. It is a quick walk through the basic concepts of Natural Language Processing. It gives an overview on text pre-processing methods like tokenization, normalization, POS (Parts of Speech). Different vectorizing techniques like modeling and word embedding are reviewed. The architectures of RNN (Recurrent Neural Network), GRU (Gated Recurrent Unit), LSTM (Long Short-Term Memory), Encoders, Decoders have been explored. It goes through latest NLP techniques like Transformers and BERT (Bidirectional Encoder Representations from Transformers).

Index Terms **- BERT, GRU, LSTM, NLP, RNN, Transformers**

1. INTRODUCTION

NLP is a branch of machine learning that involves a computer's ability to comprehend, interpret, alter, and possibly generate human language [1]. Natural Language Processing possesses the power to analyze texts and digest language more quickly than humans can comprehend what is being said or written. Understanding human language is connected to natural language processing. NLP entails educating the computer on how words connect to one another, what they mean, and what scenarios they might be used in. NLP mimics the way a human brain processes language in a machine brain. With this, text data may be organized, decoded, and understood by the machine brain.

Text translation, sentiment analysis, and speech recognition are all popular NLP applications [2]. The primary steps of NLP involve text pre-processing and vectorizing. For NLP applications RNN models are used based on the requirements. The different modifications of RNN architectures are GRU, LSTM, Encoders and Decoders, Attention Models which have their own limitations and advantages. Transformers and BERT are the most recent and efficient models.

2. TEXT PRE-PROCESSING

Text pre-processing is an NLP approach that allows machine learning models to extract structured information about a text for analysis, manipulation, or generation of new text [3]. Below are the few text pre-processing techniques which are performed on the data[4].

2.1. Corpus

A corpus is a grouping of text documents. A dataset, for example, is made up of news articles from a corpus. Twitter data containing tweets is a corpus in the same way. So, a Corpus is made up of Documents, which are made up of Paragraphs, which are made up of Sentences, and Sentences are made up of Tokens. Fig. 1 depicts all the parts of a corpus.

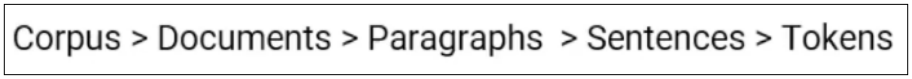

Figure 1:Corpus

2.2. Tokens

Tokens are the smallest grammatical units in a sentence or document. Words, sentences, sub words like n-grams, and characters can all be used.

2.3. N-grams

N-grams are made up of N words or letters put together. Unigrams only contain one token, but bigrams have multiple tokens in a sequence. Trigrams, on the other hand, will be made up of three tokens. In text categorization problems, N-grams are quite beneficial. Fig. 2 show the working of N-grams.

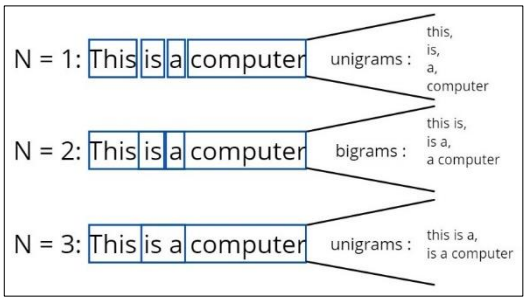

Figure 2: N-grams

Figure 3: Tokenization

2.4. Tokenization

In tokenization the phrase is divided into components which are known as tokens and this method is known a tokenization. We look for words that are composed of a series of letters. Tokenization is a basic stage in NLP processing because it helps understand the meaning of the text from its terms.

Sentence tokenization and word tokenization are the two types of tokenization procedures. Each sentence is retrieved from a paragraph in sentence tokenization, whereas each word is extracted from a sentence in word tokenization. In Fig. 3 the output of tokenization is shown.

2.5. Normalization

The process of transforming a token to its base form is known as normalization. The derivational form of a word is eliminated during the preprocessing step so that the base form can be generated. Normalization is a technique for lowering the number of unique tokens in a text and eliminating variances as well as cleaning up the text by deleting unnecessary information. Stemming and lemmatization are two typical normalization approaches.

2.6. Stemming

It's a simple technique for removing speculative forms from a token. The stem of a word is the error's output. For example, copying, copy, copies will all become cop after the stemming process. Fig.4 shows the working of stemming.

2.7. Lemmatization

The removal of a token's inflectional form and transformation into a lemma is known as lemmatization. Word structure, vocabulary, part of speech tags, and grammar relationships are all utilized. Only if the given word contains the correct parts of speech tag can lemmatization be performed. Fig.5 depicts lemmatization.

Figure 4: Stemming

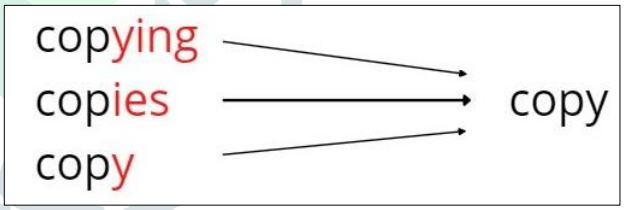

Figure 5: Lemmatization

2.8. Stop words Removal

The most prevalent words in any natural language are stop words. We eliminate them because they don't provide us with much information about our data. When it comes to interpreting data and NLP processing, stop words aren't very beneficial. The most prevalent terms in a sentence are "the," "is," "in," "for," "where," "when," "to," "at," and so on. Fig.6 show how different parts of a speech are separated and how stop words are identified.

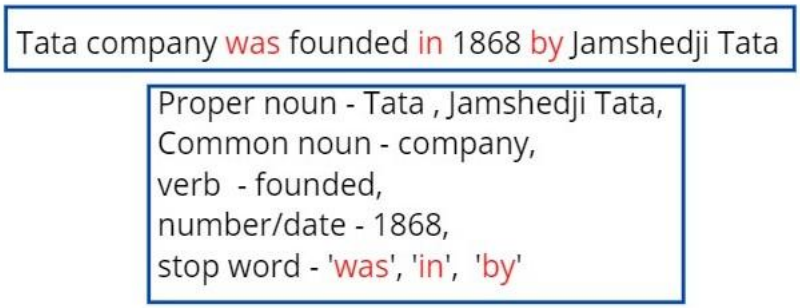

Figure 6: Stop words

2.9. Part-of-speech

POS is another NLP text preprocessing technique which deals with differentiating the terms with respect to their context and definition. We use POS to make semantic inference on the data [5]. When we deliver a sentence, for example, each word gets tagged with a noun, pronoun, verb, adjective, and so on.

3. VECTORIZING DATA

Word Embeddings involves mapping of terms and expressions from a thesaurus to a real number vector, to predict terms and semantics. Vectorization deals with turning words into numbers. Modelling techniques play a vital role in vectorizing data.

3.1. Modelling Techniques

Modelling techniques help in the identification of pattern recognition and frequency. The following are the different techniques of vectorizing data.

3.1.1. Bag-Of-Words

BOW is an NLP modeling technique where a sentence is considered as a bag consisting of words that ignore syntax [6]. It's widely used in data processing where the frequency of each term impacts on training classifiers.

The existence of words in text data is described by the Bag of Words (BoW) or Count Vectorizer. If the sentence consists of the term, then 1 is the result else 0 is the result. This gives a document matrix count for every bag of words. The working of bag of words is shown in Fig.7.

	the	red	rat	cat	eats	food
1.the red rat →	1	1	1	0	0	0
2.cat eats rat →	0	0	1	1	1	0
3.rat eats food→	0	0	1	0	1	1
4.red cat eats→	0	1	0	1	1	0

Figure 7: Bag of Words

3.1.2. TF-IDF

TF-IDF gives the significance of the term to the document among the collection of data. It is proportional to the frequency occurrence of the word in the document. Hence common terms like 'that', 'who', 'where' are ranked lower because of their lesser significance to the document.

TF-IDF for a word is calculated by multiplying the inverse document frequency of the word among groups of documents and the term frequency of the word in the document.

The term frequency is given by the frequency of the number of occurrences of the word in the document. The term frequency can later be changed based on the document's size or the raw frequency of the document's most commonly used term.

Inverse document frequency gives how recurring or less frequent a word is among the documents. The common terms are closer to zero. The TF-IDF score is proportional to the importance of the word. The formulas to calculate TF, Inverse Document Frequency, TF-IDF score are given in the Fig. 8.

$$\text{Term Frequency} = \frac{\text{Number of repetition of word in a sentence}}{\text{Total number of words in sentence}}$$

$$\text{Inverse Document Frequency} = \log\left(\frac{\text{Number of sentences}}{\text{Number of sentences containing words}}\right)$$

Term Frequency * Inverse Document Frequency

Sentences
red bus
red ball
red pen

word	frequency
red	3
bus	1
ball	1
pen	1

Sentences	feature1 red	feature 2 bus	feature 3 ball	feature 4 pen
red bus	1/2*0	1/2*0.477	0/2*0.477	0/2*0.477
red ball	1/2*0	0/2*0.477	1/2*0.477	0/2*0.477
red pen	1/2*0	0/2*0.477	0/2*0.477	1/2*0.477

Figure 8: TF-TDF

3.2. Word embedding

Word embeddings are a form of representation used to categorize words with similar meanings[7]. Word embeddings are text representations dispersed across n-dimensional space. Every word is considered as a vector consisting of real values in a space of vectors. Each word is assigned to a single vector, and the values of the vectors are found in a neural network-like manner. Fig. 9 represents word embedding in a graph.

3.2.1. Word2vec

Word2vec vectorizes the words using a neural network. Text data is taken as input and output is given by feature vectors that denote words in the data. It transforms text into a form that neural networks can comprehend. It finds similarities and groups vectors with similar words together.

The vectors produced by Word2vec give features of individual words. If given enough data, conditions, and usage, Word2vec can make fairly accurate guesses about a word's meaning based on prior occurrences. These results can be used to figure out how a word is related to other words.

Continuous Bag-Of-Words (CBOW model) and Continuous Skip-Gram model are two separate learning models that may be utilized with word2vec. Both are neural network-based architectures for learning the fundamental word representations for each word. Fig. 10 shows the working of cbow and skipgram.

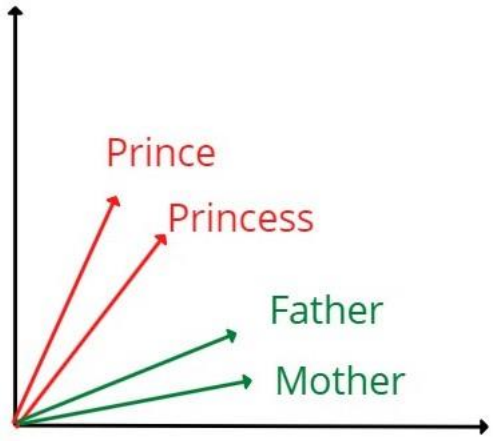

Figure 9: Word Embedding

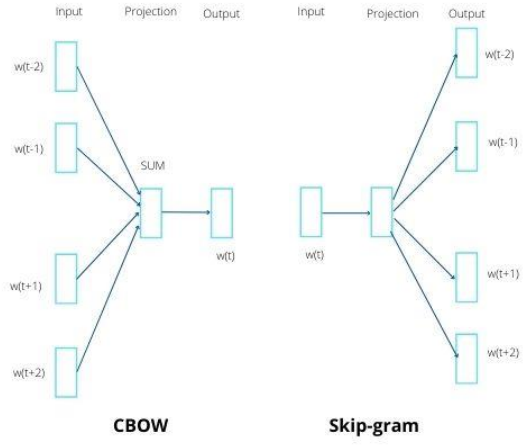

Figure 10: Cbow and Skipgram

3.2.1.1. Cbow

CBOW [8] model is constructed by forecasting the words based on its regard. A couple of terms are given as input and the model is trained to understand that the two words occur together and the same terms are added as input for that particular target word. CBOW is much faster and better than skip-gram in terms of its accuracy and training.

3.2.1.2. Skipgram

In skip-gram, the model learns by forecasting the surrounding terms and the output is predicted by considering the features of the input. If given a term, it can guess the neighbouring term and this neighbouring term will be defined by the window size. Skip-gram gives good results with huge data.

3.2.2.GloVe

Like Word2vec, Glove [9] can make fairly accurate guesses about a word's meaning based on prior occurrences. Glove uses matrix factorization approach. It involves building a big matrix with co-occurrence information, in which we count how often we see each word in a given context throughout a vast corpus.

4. RECURRENT NEURAL NETWORK

Recurrent Neural Networks are used for NLP applications. RNNs use sequential input to remember knowledge from the past [10]. When the model requires knowledge to generate an output that is dependent on the input, RNN is used. In other neural networks, all of the inputs are unrelated. However, in an RNN, all of the inputs are related to each other. The RNN recalls all of these connections during training by building networks with loops that allow it to store data. RNN has its own limitations like vanishing gradient and better architectures are proposed to overcome it. GRU, LSTM, Encoder Decoders, Attention models are different types of RNN models and Transformers and BERT are the more recent and efficient models.

4.1. Vanishing Gradient

With the increasing complexity of neural network, the gradient tends to approach zero which causes difficulty in training. As a result, the derivative's value decreases until the old and new weights' values are nearly equal.

When sigmoid function is used to activate n hidden layers, n tiny derivatives are multiplied together and when we descend to the first levels, the gradient diminishes exponentially. This causes problems with initial layers, and since these first layers are typically critical for distinguishing the essential features of the incoming data, inaccuracies throughout the entire network might result. GRU's objective is to tackle this vanishing gradient issue of RNN.

4.2. Gated Recurrent Unit – GRU

GRU is just like an LSTM unit but LSTM has an output gate whereas GRU does not. Given that both have a comparable design and occasionally yield the same outcomes. GRU can be thought of as a different variety of LSTM. GRU was primarily developed to address the vanishing gradient challenge.

GRU is made up of two parts: a reset gate and an update gate. Which data to add and which to remove is decided by the update gate. It manages the data flow into memory. The amount of data to be preserved is indicated by the reset gate. It regulates the information that is released from memory. Which information is sent to the output is determined by two vectors: the update gate and reset gate. They can be taught to retain historical knowledge or to discard data that is unrelated to the prediction.

GRU accepts input and hidden state from the previous timestamp for every timestamp t. Later, a new hidden state is produced and again given to the following timestamp. This solves the vanishing gradient problem by removing the possibility of disappearing gradients.

4.3. Long Short-Term Memory - LSTM

A recurrent neural network and an LSTM have comparable control flows [11]. As data is processed, information is sent on. The state of the cell and its gates revolve around the concept of the LSTM [12]. The cell state serves as a conduit for conveying relative information down the sequence chain. The cell state, in principle, can carry essential information all through the sequence's processing. The information flow in an LSTM cell is controlled by three distinct gates. An input-gate, a forget-gate, and an output-gate are all examples of gates.

4.3.1. Forget gate

The forget-gate tells if the data should be discarded. The sigmoid function is used which sends the data from the last hidden state and from the current input. The output will range from 1 to 0, if it is closer to 0, then data will be forgotten, if closer to 1, then it will be remembered.

4.3.2. Input gate

The input-gate updates the cell state. First, sigmoid function is implemented to combine the last concealed state as well as the current input. Network is regulated by additionally sending the hidden state and present input into the tanh function to get value in middle of -1 and 1. The sigmoid result is multiplied to tanh output. The sigmoid output determines which data of tanh will be kept.

4.3.3. Cell state

The forget vector is multiplied to the cell state pointwise. If multiplied by values close to 0, then the cell state declines. Stepwise addition is done on the outcome of the input-gate, where the update on cell state considered important by neural net. This is the new state of our cell.

4.3.4. Output gate

The last one is output gate. It identifies the next concealed state. For prediction the hidden states are also used. The current input is combined with the previous concealed state using sigmoid function and the tanh function is called with

the freshly modified cell state. By multiplying the tanh output with the sigmoid output, the information of hidden state's is determined which is the outcome. The updated concealed and cell state are then transmitted to the next time step.

4.4. Encoders and Decoders

Seq2Seq problems are handled by encoder - decoder model where the input and output pattern are of different lengths. Components of encoder and decoder are often versions of RNN, such as GRU or LSTM. This is due to their ability to capture long-term dependencies despite the vanishing gradient problem. Fig. 11 depicts the 2 blocks of encoder and decoder.

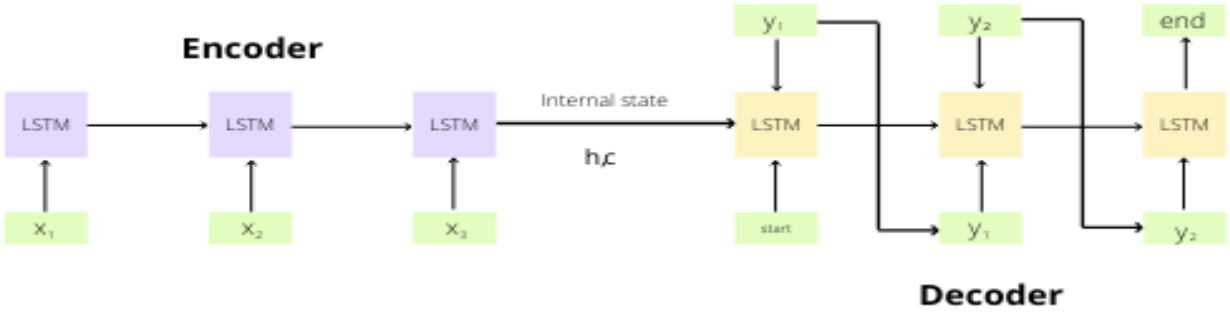

Figure 11: Encoder and Decoder

4.4.1.1. Encoder

Input sequence is interpreted by an encoder LSTM, with one word supplied into the encoder at each timestep. The data is then processed at each timestep, and the relevant knowledge in the input sequence is captured. Initialization of decoder happens using the cell and hidden state of previous time.

4.4.1.2. Decoder

A decoder is programmed to infer the upcoming term in the sequence. It is a kind of LSTM network that analyses the whole anticipated sequence word to word and deduces a sequence that is one time step delayed.

Before the intended sequence is delivered to the decoder, tokens like 'start' and 'end' are inserted to it. The intended sequence is unavailable while deciphering the test sequence. As a result, we begin anticipating the desired sequence by feeding the decoder the first word, which is invariably the 'start' token.

The 'end' marker indicates that the statement has come to an end.

4.4.2. Seq2Seq

To handle language challenges seq2seq model is used which is a type of RNN. When the length of the input and output to the model differs, the sequence-to-sequence approach tries to map input text with set length to output text with set length. The input and output are both sequences in seq2seq [13].

Encoder-decoder is the most common design for Seq2Seq models. The encoder turns the input into a constant vector, which the Decoder then interprets and outputs. It works fine for short sequences, but it fails when we have a long sequence since the encoder finds it impossible to memorize the full sequence into a fixed-sized vector and compress all of the contextual information. As the length of the sequence grows longer, the model's performance begins to deteriorate. We employ attention models to get over this. Fig. 12 shows the working of sequence-to-sequence model.

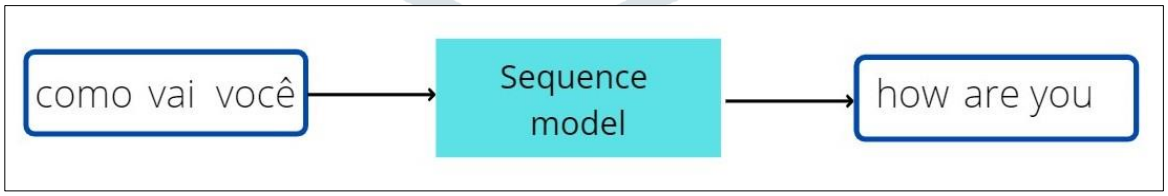

Figure 12: Sequence to Sequence

4.5. Attention Models

In the attention mechanism, information is collected from each time step of the encoder[14][15]. The weighting is determined by how important that time step is for the decoder to create the next word in the sequence optimally. When the model predicts an output word, it just looks at areas of the input where the most relevant information is concentrated, rather than the full phrase, and it only looks at a few input words. The encoder functions normally, and the only variation is in the decoder. A context vector, the previous output, and the prior hidden state are used to calculate the decoder's hidden state.

Input is received from each time step of the encoder to implement the attention mechanism, but importance is given to the time steps. The weighting is determined by how important that time step is for the decoder to create the next word in the sequence optimally. The relevance of certain parts of the source sequence can be raised to result in the target sequence rather than checking all the terms in the source sequence. The attention mechanism is based on this concept. Fig. 13 shows the attention model.

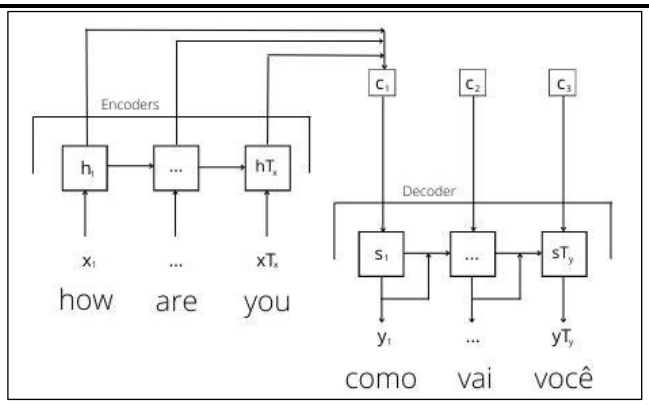

Figure 13: Attention model

The three types of Attention possible in a model:

1. Encoder-Decoder Attention: Special attention is given to output and input pattern.
2. Self-attention in the input sequence: All the terms in the input sequence are taken into consideration.
3. Self-attention in the output sequence: Self attention is only limited to the words that come before a certain word. This ensures that no information is leaked during the model's training. For each stage, this is accomplished by hiding the words that follow it. So, the first term of the output sequence is not hidden in step 1, the first two terms are not hidden in step 2, and so on.

4.6. Transformers

Transformer is built on a one-of-a-kind mechanism that manages both long-range interactions and sequence-to-sequence tasks. Transformer[16][17] is a two-part design that transforms one sequence into some other without using Recurrent Networks.

One is the encoder and the other one is the decoder. Each of them is composed of units that can be layered on top of one another multiple times. Multi-Head Attention and Feed Forward layers make up the units. Only embedded n-dimensional space inputs or outputs are used because explicit strings cannot be used. Crucial aspect of the model is positional encoding. Depending on the order of its constituents each word is assigned a relative position and there is no recurrent network that can recall the sequences on how the model is fed.

4.7. BERT

A crucial aspect of BERT [18] is positional encoding of different words. Each and every term in the sequence is given a position since we don't have any network that can remember the sequence in which data is fed so it depends on the sequence of its members.

BERT uses Transformer which is an attention mechanism which learns relevant associations between terms in a phrase. Transformers is formed of two distinct approaches, one where text is given as input to encoder and next where task inference is done by decoder. To create a language model BERT requires encoders approach.

The encoder goes through the complete series of terms at one go, instead of reading the text input one word at a time. As a result, it's referred to as bidirectional, however it's more accurate to refer to it as non-directional. This feature helps the model to interpret a term's context from its surrounding terms.

5. Conclusion

NLP is a booming field with a lot of room for research and advancement. Various NLP strategies have been described in this work, ranging from basic procedures and terminologies to contemporary breakthroughs. The recent architectures like BERT and Transformers are proved to be more efficient and useful than traditional NLP models. While it is simple for people to learn a language, NLP is challenging for machines to implement due to the ambiguity and imprecise nature of natural languages. Reading, understanding, and comprehending human languages in a useful way is the main goal of NLP.

6. REFERENCES

[1] Ranjan, Nihar, et al. "A Survey on Techniques in NLP." International Journal of Computer Applications 134.8 (2016): 6-9.

[2] Degefa, Shambel Dechasa, Oromia Batu, and Bekele Abera Hordofa. "A Review of Natural Language Processing Techniques: Application to Afan Oromo." *Phonology* 1.1: 2.

[3] Reshamwala, Alpa, Dhirendra Mishra, and Prajakta Pawar. "Review on natural language processing." *IRACST Engineering Science and Technology: An International Journal (ESTIJ)* 3.1 (2013): 113-116.

[4] Chowdhary, KR1442. "Natural language processing." *Fundamentals of artificial intelligence* (2020): 603-649.

[5] Behera, Santosh K., and Mitali M Nayak. "Natural Language Processing for Text and Speech Processing: A Review Paper." *International Journal of Advanced Research in Engineering and Technology (IJARET)* (2020).

[6] Juluru, Krishna, et al. "Bag-of-words technique in natural language processing: a primer for radiologists." *RadioGraphics* 41.5 (2021): 1420-1426.

[7] Almeida, F., Xex´eo, G.: Word embeddings: A survey. arXiv preprint arXiv:1901.09069 (2019)

[8] Li, Teng, et al. "Contextual bag-of-words for visual categorization." *IEEE Transactions on Circuits and Systems for Video Technology* 21.4 (2010): 381-392.

[9] Sharma, Yash, et al. "Vector representation of words for sentiment analysis using GloVe." *2017 international conference on intelligent communication and computational techniques (icct)*. IEEE, 2017.

[10] Tarwani, Kanchan M., and Swathi Edem. "Survey on recurrent neural network in natural language processing." *Int. J. Eng. Trends Technol* 48.6 (2017): 301-304.

[11] Yu, Yong, et al. "A review of recurrent neural networks: LSTM cells and network architectures." *Neural computation* 31.7 (2019): 1235-1270.

[12] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[13] Chung, Yu-An, and James Glass. "Speech2vec: A sequence-to-sequence framework for learning word embeddings from speech." *arXiv preprint arXiv:1803.08976* (2018).

[14] Niu, Z., Zhong, G., Yu, H.: A review on the attention mechanism of deep learning. Neurocomputing 452, 48–62 (2021)

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems 30 (2017)

[16] Braşoveanu, Adrian MP, and Răzvan Andonie. "Visualizing transformers for nlp: a brief survey." 2020 24th International Conference Information Visualisation (IV). IEEE, 2020.

[17] Wolf, Thomas, et al. "Transformers: State-of-the-art natural language processing." Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. 2020.

[18] Tenney, Ian, Dipanjan Das, and Ellie Pavlick. "BERT rediscovers the classical NLP pipeline." arXiv preprint arXiv:1905.05950 (2019).