Emotion Classification of Twitter Data using Classical and Deep Learning Methods

Jianxiao Yang yangjx@bu.edu
M. Srinivasan msrini@bu.edu
Wenjun Zhang wjz@bu.edu

Abstract

This study focuses on exploring and comparing the effectiveness of traditional machine learning techniques and advanced deep learning approaches in the context of text classification. We employed conventional methods such as Logistic Regression and Support Vector Machines (SVM), alongside deep learning strategies including BERT embeddings, LoRA fine-tuning, Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNN). The performance of these models was rigorously evaluated on a comprehensive dataset. Our findings indicate that models based on BERT embeddings significantly outperform traditional machine learning approaches, highlighting the superiority of deep learning techniques in handling complex text classification tasks.

1. Introduction

Text classification is a fundamental task in natural language processing, involving the categorization of textual data into predefined classes. This study aims to investigate the efficacy of various machine learning and deep learning methodologies in text classification. Traditional machine learning methods, such as Logistic Regression and SVM, have been widely used in this domain due to their simplicity and efficiency. However, with the advent of deep learning, more sophisticated approaches like BERT embeddings, LoRA fine-tuning, LSTM, and CNN have emerged, offering enhanced feature extraction and learning capabilities.

Our experiment leverages both traditional and deep learning methods to develop a comprehensive text classification model. The traditional approaches provide a baseline for performance comparison, while the deep learning techniques, particularly those based on BERT, are hypothesized to offer superior performance due to their advanced contextual understanding and adaptability. The objective of this analysis is to evaluate these methodologies on a standardized dataset, aiming to reveal the most effective strategies for text classification in the current technological landscape.

2. Literature review

The field of natural language processing (NLP) has witnessed substantial evolution, especially in semantic understanding and text classification. Historically, various traditional embeddings xsand machine learning methods have been pivotal in advancing these tasks.

Early approaches in semantic representation primarily utilized word-based embeddings such as Bag-of-Words (BoW) [1] and Term Frequency-Inverse Document Frequency (TF-IDF)[2]. These techniques represent texts as vectors of word counts or frequencies, providing a basic yet effective way of capturing semantic information. However, they often fail to encapsulate the context and word order, which are crucial for understanding nuanced language semantics.

The introduction of distributed representations marked a significant advancement. Word2Vec [3], developed by Mikolov et al., and GloVe [4] by Pennington et al., are notable examples. These models generate embeddings that capture semantic similarities by representing words as dense vectors in a continuous vector space. This approach led to a better grasp of word relationships but still lacked the context sensitivity for entire sentences or documents.

Traditional machine learning techniques, including Naive Bayes [5], Decision Trees [6], and Random Forests [7], have been extensively applied in text classification. These methods, when combined with word embeddings, provided a baseline for semantic analysis. However, the most prominent were Support Vector Machines (SVM) [8] and Logistic Regression [9], known for their effectiveness in high-dimensional spaces typical for text data.

Despite the progress, these traditional methods had limitations. They often required extensive feature engineering and struggled with capturing long-term dependencies and context in language. The advent of deep learning and neural network-based models, such as Long

Short-Term Memory (LSTM) [10] networks and Convolutional Neural Networks (CNN) [11], started to address these challenges, leading to more nuanced and context-aware semantic understanding. Developed by Google, BERT [12] revolutionized the field of NLP by introducing a transformer-based model that captures bidirectional contexts, allowing for a deeper understanding of language semantics. Unlike traditional embeddings, BERT considers the entire context of a word by looking at the words that come before and after it. This has been a significant leap from previous models that typically processed text in a unidirectional manner.

BERT and its variants, such as RoBERTa [13], DistilBERT [14], have demonstrated state-of-the-art performance in various NLP tasks, including semantic analysis and classification. These models are pre-trained on large corpora and can be fine-tuned for specific tasks, which allows them to achieve exceptional accuracy even with smaller labeled datasets. The effectiveness of BERT in capturing nuanced linguistic features has made it a preferred choice for complex NLP tasks.

The integration of BERT and transformer-based models into NLP practices marks a pivotal shift from traditional embeddings and machine learning methods to more advanced, context-aware approaches. This transition underlines the field's ongoing commitment to developing more sophisticated and effective tools for semantic understanding and classification

3. Dataset and loss

3.1. Dataset

In our research, we are utilizing the dataset from https://huggingface.co/datasets/dair-ai/emotion, which is derived from the GoEmotions dataset, extensively known for its comprehensive coverage of emotional expressions in text. This dataset is structured for multi-label classification, where each text entry is tagged with one or more emotion categories, adding a layer of complexity to our text classification task. Our focus is to develop a model capable of discerning and categorizing the emotional content in textual data, which is crucial in applications like sentiment analysis, customer feedback, and social media monitoring. The diversity in the source material and the range of emotions represented in this dataset provides us with a rich resource for training and testing our model, enabling us to capture the nuanced emotional undertones inherent in natural language.

3.2. Loss and evaluation

In evaluating our multi-class text classification model, we have implemented a comprehensive set of metrics to accurately gauge its performance.

For our loss function, we utilize the multi-class cross-entropy loss, also known as categorical cross-entropy. This loss function is well-suited for multi-class classification problems, where the output could belong to one among many possible categories. The cross-entropy loss for multi-class classification is defined as:

$$L = -\sum_{i=1}^{M} y_i log(\hat{y}_i) \#(1)$$

is the predicted probability for that class. The goal during training is to minimize this loss, driving the model to predict a probability distribution over the classes that is as close as possible to the true distribution.

By employing these metrics and loss function, our evaluation framework ensures a thorough and discriminating analysis of the model's capability to classify text into one of several categories, which is critical for the success of our classification objectives.

4. Experimental results

4.1. Methodology and Parameters

In our approach to developing a robust multi-class text classification model, we adhered to a straightforward yet effective methodology. We utilized the predefined splits of the dataset, comprising training, validation, and test subsets.

For the training process, we utilized Machine learning and deep learning ways.

4.2. Machine Learning training

In training our classical Machine Learning models, we adopted a systematic approach that involved several key steps, leveraging both Natural Language Processing (NLP) techniques and machine learning algorithms. Following Random Forest and SVM, We utilized the **nltk** library to download necessary components such as 'punkt' for tokenization and 'stopwords' to filter out common English words that typically don't contribute to text meaning.

Data was sourced from **JSONL** files (for training, validation, and testing) and preprocessed using tokenization, lowercasing, and stopword removal. This preprocessing transformed the raw text into a more analyzable form.

We used the **TfidfVectorizer** from the **sklearn.feature_extraction.text** module to convert the preprocessed text data into numerical feature vectors. This method assigns a numerical weight to each word based on its frequency and inverse document frequency, which helps in assessing the importance of words across the dataset. The vectorizer was fitted to the training data and

then used to transform both the training and unseen validation/test data.

SVM Classifier Training:

For the SVM model, in order to aim at the performance, instead of reimplementation, we mainly use SVC in the **sklearn** library to train, we chose the Radial Basis Function (RBF) kernel, linear, and polynomial, with a regularization parameter=0.5, *C*=0.5. This choice suggests a balance between training data fit and generalization.

Random Forest Classifier Training:

For the same reason above, in order to aim at the performance, instead of reimplementation, we mainly use SVC in the **sklearn** library to train. Because Random Forest is an ensemble learning method known for its robustness and effectiveness in handling various types of data, only following two parameters will affect its result in big differences, depth and num_trees. It is initialized with 100 decision trees and a fixed random state for consistency in results.

We trained the SVM/Random Forest using the vectorized training data (X_train) and corresponding labels (y_train). Post-training, the model's performance was evaluated on both validation and test data sets. We used metrics like accuracy and a detailed classification report to understand the model's effectiveness in classifying the data correctly.

K Nearest Neighbour Classifier Training:

KNN, a non-parametric and instance-based learning algorithm, functions on the principle of similarity, where instances are classified based on the majority class among their neighboring data points.

The KNeighborsClassifier from the sklearn library was implemented to train the KNN model. Hyperparameter tuning was performed using GridSearchCV to determine the optimal number of neighbors (k) for the algorithm.

GridSearchCV was leveraged to identify the optimal k value through cross-validation, ensuring the model's robustness and accuracy.

Logistic regression for Multi-class Classification:

Logistic Regression for multi-class classification is a statistical technique extending from binary to multiple classes, employing the softmax function to predict the probabilities of an input belonging to each class in a set of several categories.Utilizing the sklearn library's LogisticRegression module, this method was employed to train the model. Hyperparameter configuration, such as the 'lbfgs' solver for optimization and 'multinomial' designation for multi-class classification, ensured an effective fit for handling textual data. L2 regularization is also incorporated in LogisticRegression module which mitigates overfitting and enhances the generalization capacity of the model.

4.3. Deep learning training

We utilized the 'bert-base-uncased' pre-trained model from Hugging Face as the embedding layer for our deep learning architecture, restricting BERT to just two layers of transformer encoders. This streamlined BERT model forms the foundation of our system, to which we appended various neural network structures—namely, CNN, CNNHeavy, and LSTM—each forming a separate model variant for end-to-end training. We began our training process by setting epochs to four and fine-tuning the batch size and learning rate to identify parameter settings that yield promising results. The optimal parameters were found to be a batch size of 64 and a learning rate of 0.0001. With these parameters, we proceeded to train each model variant for eight epochs, while also experimenting with varying numbers of CNN layers to assess their impact on the model's performance.

We also employed the full BERT model, utilizing all 12 layers of transformer blocks, and opted not to append any additional blocks. Instead, we performed fine-tuning using the Low-Rank Adaptation (LoRA) technique, without integrating other architectural elements like CNN or LSTM blocks and only fine-tuning the Key Value matrix in the attention block. We fine tune the rank and alpha from 8 to 64.

5. Performance

5.1. Machine Learning Result

SVM and Forest result are in:

https://github.com/esbqjl/text_classification/tree/main/result

The accuracy of SVM with a linear kernel is the highest, which we did not expect. We first assume the dataset will be kinda massive in some point, however, emotion sets between each other obtain good linear separability. It has up to 82.2% accuracy, in some experiments, it can be reached up to 90% accuracy. The overwhelming performance of the linear kernel makes the other three kernels less pretty, especially for polynomials; it has only 45.85% accuracy in the validation set. In this case, soft margin is considered already, which C is set by 0.5.

Random forest has around 89.15% accuracy no matter how many trees. It has a stable performance due to its mechanism. It is a good model but not a God model, based on the complication of the dataset, but still, when facing a low dimension/ light dataset, it always is a standard indicator to compare with other models.

Logistic Regression exhibits notable accuracy, particularly in scenarios where the textual data showcases

linear separability between emotional categories. The model achieves an accuracy of up to 86.4%, even reaching 90% in select experiments.

In contrast, K-Nearest Neighbors (KNN) demonstrates competitive performance with its distinctive mechanism based on instance-based learning. Though lacking the linear model's delineation, KNN maintains its versatility and achieves consistent accuracy of 80.4%, although below that of the linear kernel's performance. The KNN model presents reliable results, showcasing a stable accuracy level that remains a benchmark in scenarios involving low-dimensional or lightweight datasets

One thing needed to be brought up, in the realm of Natural Language Processing (NLP), a key advancement lies in the translation of texts into training datasets. For this purpose, TF-IDF (Term Frequency-Inverse Document Frequency) is a well-established technique. It functions as a classical text translator in machine learning, converting texts into a metric that reflects word significance based on their frequency and uniqueness across documents. Nonetheless, the field offers a plethora of alternative methods that could potentially elevate performance. These include advanced techniques like encoders and embeddings, predominantly rooted in deep learning, yet compatible with traditional machine learning frameworks. While TF-IDF generates a metric embedded with an Inverse Document Frequency (IDF) score, the latter techniques produce a matrix representation of data. Both methodologies are effective, particularly when the input dataset is structurally aligned with the target labels, enabling more nuanced and accurate machine learning models. A crucial aspect to highlight is the nuanced handling of textual meaning by encoder and embedding methods, as opposed to the straightforward relevance metrics derived from TF-IDF. These contemporary techniques delve deeper into the semantics of words within their specific contexts, capturing intricate relationships in the text. This capability is especially valuable in dealing with complex linguistic phenomena such as urban slang or memes, where the conventional relevance value assigned by a dictionary may fall short. In summary, while TF-IDF offers a foundational approach to text translation in NLP, encoders and embeddings provide a more sophisticated, context-aware analysis, enhancing the ability to interpret and process the multifaceted nature of human language.

Here is the link to the K Nearest Neighbors and Logistic Regression for Multi-class Classification:

https://github.com/ManasviniSrini/TextClassification_KNN_LR

The Bayes result is in:

https://github.com/Yangjianxiao0203/text-classification/blob/cloud/eval/bayes_eval.json.

The accuracy only reached 76.6% with poor recall rate(53.9%). Primarily, the Bayes model assumes feature independence, which may not hold true in complex text data where context and word relationships are essential. This naive assumption can lead to misclassification of instances that rely on contextual clues, thus impacting recall.

Moreover, the Bag-of-Words (BoW) representation used for feature extraction doesn't capture word order or semantics, which can limit the model's ability to differentiate between classes that share similar vocabulary but differ in syntax or composition. This limitation could result in a higher number of false negatives, directly reducing the recall rate.

Additionally, the Multinomial Naive Bayes model may not handle the imbalanced class distribution well if the dataset has varied class frequencies. In such cases, the model might be biased towards the more frequent classes, thus failing to identify instances of less frequent ones, further decreasing the recall.

5.2. Deep Learning Result

The bert based results are in:

https://github.com/Yangjianxiao0203/text-classification/blob/cloud/result.csv.

The base model's performance is 84% with 12 layers. The results are shown in table 1. We draw conclusion that:

(1) Fine-tuning BERT has proven highly effective. Due to computational resource constraints, we employed just two layers of BERT and adjusted for only eight epochs, resulting in an increase in accuracy from 84% to 91%.

(2) Adding an LSTM layer post-BERT did not yield a significant performance boost but substantially increased training time. The increase in computational time is attributed to the LSTM's inherent sequential nature, which, unlike attention mechanisms, cannot be parallelized, leading to inefficiencies. Moreover, the lack of performance gain is likely because LSTMs essentially serve as a weaker version of attention by capturing previous tokens' information, thus offering no substantial enhancement to BERT.

(3) Augmenting BERT with a CNN, specifically a three-layer CNN combo: convolution, normalization, activation, and dropout layers, effectively mitigated overfitting and increased accuracy by 0.7%. This improvement is possibly because CNNs inherently focus on local information, acting like a sliding window. Since

many reviews contain phrases with strong sentiment indicators, the model's global attention is complemented by the CNN's emphasis on localized features, akin to selective dropout, thereby enhancing model accuracy.

Table 1. Bert based model and bayes performance

model_type	epoch	num_layers	max_length	hidden_size	batch_size	learning_rate	acc	f1_score	recall	precision
bayes							0.7655	0.5632	0.539	0.6938
bert_cnn	8	1	128	128	32	0.0001	0.9225	0.90159402	0.90768066	0.89740877
bert_cnn	8	1	256	128	32	0.001	0.275	0.07189542	0.16666667	0.04583333
bert_cnn	8	1	64	128	64	0.0001	0.9205	0.90238055	0.91413984	0.89212396
bert_cnn	8	1	128	128	64	0.001	0.644	0.4817138	0.49194955	0.61564719
bert_cnn	8	1	64	128	64	0.001	0.6285	0.41992883	0.46034973	0.4244426
bert_cnn	8	1	64	128	32	0.0001	0.9215	0.89562927	0.88725868	0.90984662
bert_cnn	8	1	256	128	64	0.001	0.6655	0.43580949	0.46633916	0.42196902
bert_cnn	8	1	64	128	32	0.001	0.6605	0.46086922	0.48714394	0.48655645
bert_cnn	8	1	256	128	32	0.0001	0.8925	0.86524476	0.8673188	0.86440605
bert_cnn	8	1	128	128	32	0.001	0.6055	0.42391803	0.4466917	0.46025582
bert_cnn	8	1	128	128	64	0.0001	0.9175	0.88837817	0.8914743	0.88877113
bert_cnn_heavy	8	1	64	128	64	0.0001	0.9185	0.88307961	0.89809584	0.87193744
bert_cnn_heavy	8	2	64	128	64	0.0001	0.9165	0.86715676	0.86901906	0.86585504
bert_cnn_heavy	8	3	64	128	64	0.0001	0.9185	0.88517816	0.90888831	0.87017325
bert_cnn_heavy	8	1	64	128	32	0.0001	0.9145	0.87282163	0.86942068	0.88144956
bert_cnn_heavy	8	2	64	128	32	0.0001	0.917	0.88410096	0.88341638	0.88579622
bert_cnn_heavy	8	3	64	128	32	0.0001	0.924	0.88071501	0.87931439	0.88410559
bert	8	1	64	128	64	0.0001	0.9175	0.87386671	0.87768438	0.8712885
bert	8	1	64	128	32	0.0001	0.9115	0.87427631	0.89747078	0.86205709
bert_lstm	8	1	64	128	64	0.0001	0.912	0.86544084	0.85936497	0.87230853
bert_lstm	8	2	64	128	64	0.0001	0.912	0.87029464	0.86918246	0.87400003
bert_lstm	8	3	64	128	64	0.0001	0.915	0.87487779	0.86624251	0.88579218
bert_lstm	8	1	64	128	32	0.0001	0.915	0.87584651	0.87694388	0.87817639
bert_lstm	8	2	64	128	32	0.0001	0.915	0.87418663	0.87030043	0.8785967
bert_lstm	8	3	64	128	32	0.0001	0.9185	0.88118253	0.86836998	0.89782468

Analyzing the performance of the full BERT model fine-tuned with LoRA only in attention(Key, Value), as reflected in the provided table 2 , we see varying degrees of accuracy, F1 score, recall, and precision across different configurations of LoRA's hyperparameters lora_r and lora_alpha. We draw conclusion that:

(1) Generally, as lora_r and lora_alpha increase, we see an improvement in all performance metrics. This suggests that allowing more parameters (higher rank) and a larger adaptation effect (lora_alpha) leads to better model performance up to a certain point.

(2) The best performing model in terms of accuracy, F1 score, and precision has lora_r set to 8 and lora_alpha set to 64 with 92.5% accuracy. This combination seems to provide a good balance between model capacity and the ability to adapt to the specific task.

(3) Increasing lora_r from 8 to 16, then to 32, does not consistently improve performance when lora_alpha is held constant. For example, with lora_alpha set to 8, increasing lora_r actually decreases accuracy and F1 score.

(4) The model with lora_r set to 16 and lora_alpha set to 64 seems to achieve the highest recall but not the highest precision, suggesting a potential trade-off between these metrics.

(5) The entries with lora_r set to 32 and lora_alpha set to 64 do not show a significant variation in performance across the metrics compared to lower ranks, which may indicate that further increasing the rank does not contribute to the model's learning capacity given the current setup or task complexity.

Table 2. bert(12 layers) fine tune with lora

Lora_r	lora_alpha	acc	f1_score	recall	precision
8	8	0.8375	0.66066391	0.6584411	0.68737844
8	16	0.9015	0.84125269	0.82849071	0.86834246
8	32	0.887	0.8123212	0.81232832	0.84203259
8	64	0.925	0.88268323	0.88175857	0.88823195
16	8	0.8365	0.62501228	0.6346954	0.68026017
16	16	0.9075	0.82382759	0.81518769	0.87362642
16	32	0.915	0.86775182	0.88309161	0.86012446
16	64	0.919	0.87786487	0.89257641	0.86787581
32	8	0.854	0.67374908	0.67372506	0.71307807
32	16	0.905	0.79491649	0.79245561	0.89424016
32	32	0.917	0.87078629	0.86982418	0.87914259
32	64	0.9075	0.85604004	0.84643424	0.88927733

By comparing LoRA and Small model methods, we draw conclusion that:

(1) Utilizing LoRA, even with a 12-layer BERT, the number of parameters adjusted remains fewer than that of a 2-layer BERT. Yet, its performance surpasses that of a 2-layer BERT enhanced with CNN. This suggests that, assuming a similar number of adjustable parameters, fine-tuning the most critical layers (Key, Value) is more impactful than downsizing the model and applying full-scale fine-tuning.

(2) The attention mechanism itself is sufficiently sophisticated. Merely fine-tuning the Key and Value can significantly enhance model performance, indicating that the attention mechanism is indeed key to the model's understanding of semantics.

(3) The Small Model concept is valid, and with the application of a CNN-heavy approach, the model's performance closely matches that of a 12-layer BERT fine-tuned with LoRA.

6. Deploy

We deploy the model into onnx and we build a tiny backend and frontend page for it. Please git clone https://github.com/Yangjianxiao0203/text-classification.git and switch to the onnx branch.

Run python app.py and the page will be shown in localhost: 5001.

7. Future work

In future work, we aim to enhance text classification efficacy by delving into the experimental direction of hybridizing attention mechanisms within BERT to specialize in distinctive linguistic features. We plan to explore data augmentation to bolster the training set and transfer learning techniques for domain-specific tuning. A significant thrust will be towards developing cross-lingual capabilities, enabling the model to operate over diverse languages. Additionally, we will prioritize model interpretability and the integration of structured data to enrich contextual understanding. Advanced hyperparameter optimization and active learning strategies will be employed to refine model performance further, and we will consider the potential benefits of employing Neural Architecture Search (NAS) to autonomously discover more efficient model architectures, moving towards a more adaptive, efficient, and transparent text classification system.

8. Conclusion

In comparing Machine Learning and Deep Learning for text classification, particularly in emotion analysis, we observe distinct advantages and limitations. Machine Learning models like SVM with a linear kernel have demonstrated high accuracy, reaching up to 90%, and ensemble methods such as Random Forest have shown stable performance with around 89.15% accuracy. They are also commended for their computational efficiency and interpretability. However, their limitations become apparent when handling complex, non-linear data, with polynomial kernel accuracy dropping to 45.85%. The assumption of feature independence can also lead to lower recall rates, and they generally struggle with capturing context and semantic relationships, as seen with Naive Bayes models, which have an accuracy of 76.6%.

On the other hand, Deep Learning, with methods like BERT fine-tuning, excels in understanding complex patterns, achieving 91% accuracy. These models are capable of capturing both local and global dependencies in data, evidenced by a CNN's slight improvement of BERT by 0.7%. Their advanced handling of context and

semantics makes them powerful for language tasks. However, this comes at the cost of higher computational complexity and resource requirements. Moreover, adding additional layers like LSTM doesn't always yield performance improvements and can extend training times. Nonetheless, fine-tuning critical layers, such as with LoRA, is necessary to achieve significant improvements in complex tasks.

9. Individual effort

Jianxiao Yang: mainly focus on deep learning, bert-related models, bayes model, deployment(frontend/backend/onnx), leverage AWS, build whole deep-learning pipeline for training and deploying

Manasvini Srinivasan: Logistic Regression, KNN

Wenjun Zhang: mainly focus on machine learning, SVM, Random Forest, local implementation, accuracy comparison.

References

[1] Jégou Hervé, Douze Matthijs, and Schmid Cordelia. Improving Bag-of-Features for Large Scale Image Search. *International Journal of Computer Vision*, 83(3):316-336, 2010. DOI: doi.org/10.1007/S11263-009-0285-2

[2] Shahzad, Qaiser., Ramsha, Ali. (2018). Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents. International Journal of Computer Applications, doi: 10.5120/IJCA2018917395

[3] Tomas, Mikolov., Kai, Chen., Greg, S., Corrado., Jeffrey, Dean. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv: Computation and Language,

[4] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[5] Irina, Rish. (2000). An empirical study of the naive Bayes classifier.

[6] Rodrigo, C., Barros., André, C., P., L., F., de, Carvalho., Alex, A., Freitas. (2014). Decision-Tree Induction. doi: 10.1007/978-3-319-14231-9_2

[7] Leo, Breiman. (2001). Random Forests. doi: 10.1023/A:1010933404324

[8] Corinna, Cortes., Vladimir, Vapnik. (1995). Support-Vector Networks. Machine Learning, doi: 10.1023/A:1022627411411

[9] A., J., Scott., David, W., Hosmer., Stanley, Lemeshow. (1991). Applied Logistic Regression.. Biometrics, doi: 10.2307/2532419

[10] Alex, Sherstinsky. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. Physica D: Nonlinear Phenomena, doi: 10.1016/J.PHYSD.2019.132306

[11] Asifullah, Khan., Anabia, Sohail., Umme, Zahoora., Aqsa, Saeed, Qureshi. (2020). A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review, doi: 10.1007/S10462-020-09825-6

[12] Ashish, Vaswani., Noam, Shazeer., Niki, Parmar., Jakob, Uszkoreit., Llion, Jones., Aidan, N., Gomez., Lukasz, Kaiser., Illia, Polosukhin. (2017). Attention is All you Need.

[13] J., Briskilal., C., N., Subalalitha. (2021). An ensemble model for classifying idioms and literal texts using BERT and RoBERTa. Information Processing and Management, doi: 10.1016/J.IPM.2021.102756

[14] Matteo, Muffo., Enrico, Bertini. (2023). BERTino: An Italian DistilBERT model. 2023. doi: 10.4000/books.aaccademia.8748

[15] Minjae Kim, Seho Son, Ki-Yong Oh, "Margin-Maximized Hyperspace for Fault Detection and Prediction: A Case Study With an Elevator Door", IEEE Access, vol.11, pp.128580-128595, 2023.

[16] Gilles Louppe, 28 Jul 2014, Understanding Random Forests: From Theory to Practice arXiv:1407.7502 [stat.ML] https://doi.org/10.48550/arXiv.1407.7502

[17] Gerard, Salton., Chris, Buckley. (1988). Term Weighting Approaches in Automatic Text Retrieval. Information Processing and Management, 24(5):323-328. doi: 10.1016/0306-4573(88)90021-0NLTK: The Natural Language Toolkit (Bird & Loper, ACL 2004)