COMPARISON BAGGING AND SUPPORT VECTOR MACHINE FOR CLASSIFICATION SOFTWARE REQUIREMENT

[1]Klaus Rajendra Wastu
[1]Program Studi Teknik Informatika Fakultas Ilmu Komputer, Universitas Katholik Soegijapranata
[1]20k10036@unika.ac.id

Abstract

Software Requirements Specifications is a document that describes the requirements that occur in the development of a software system. The category of requirements is defined in two types: Functional Requirements (FR) and Non-Functional Requirements (NFR). Software Requirements Engineering is critical in successfully designing a piece of software. Many studies have examined the classification of software requirements using machine learning, but none have compared bagging algorithms with Support Vector Machine (SVM). This study compares text feature extraction techniques with machine learning algorithms Bagging and Support Vector Machine to solve the Software Requirement Classification problem. Using vectorization techniques from word2vec: Continuous Bag of Words and Skip-gram can help produce the best model performance for Bagging and SVM models. In this study, the data used is expansion data from the PROMISE repository, namely PROMISE_exp, the repository is a collection of software requirements data that has been labeled. To measure performance, this study uses an evaluation matrix, namely precision, recall and f1-score. As a result, the two models that have been trained using the Continuous Bag of Words and skip-gram vectorization techniques will be compared to determine the more optimal model for classifying software requirements from the promise_exp repository.

Keywords: CUDA, GPU, CPU, Parallel / Bagging, Word2vec, Support Vector Machine, Software Requirement, Machine Learning

Introduction

Software Requirements Specifications is a document that describes the requirements that occur in the development of a software system. The category of requirements is defined in two types: Functional Requirements (FR), drawing the functions that the system provides, and Non-Functional Requirements, the limitations of the developed application [1]. Software requirements engineering is critical in successfully designing a piece of software. Because of this, many software system developers fail because they ignore non-functional requirements. The task of software requirements classification is to categorize software requirements documents into functional requirements and non-functional requirements [1].

There have been many studies on the classification of software requirements into functional requirements and non-functional sub-classes. And the algorithms that are often used are Support Vector Machine and Naive Bayes. This research will compare the Support Vector Machine algorithm which is widely used in the classification process compared to the Bagging algorithm. According to Arfiani and Rustam [2], the Bagging algorithm is one of the new and most successful computational methods for clustering large and unstable data. The software documentation used in this research is the PROMISE_exp dataset. This research will compare Support Vector Machine algorithm with Bagging algorithm in software requirement classification on PROMISE_exp dataset. With the feature extraction technique / vektorization technique of word2vec using Continuous Bag of Words and Skip-gram [3], it can help to improve the performance of the model.

This research will compare the accuracy of 2 algorithms that will be compared, namely Bagging and Support Vector Machine. Later the results of the classification of software requirements made will help developers to classify software requirements documentation into appropriate categories.

Research by Zahra et al [3] aimed at improving the automatic classification of requirements into FR and NFR by using pre-processing techniques such as POS Tagging. They achieved a 4.48% improvement in classification using the C4.5 decision tree algorithm. However, these results differ from the study by Edna and Bruno [1], which states that SVM is better than Naive Bayes. Zijad and Walid's research [4] supports the use of SVM for requirements classification with good results. While Sebastiano et al's research [5] focused on the classification of application user reviews using the J48 algorithm. Mengmeng Lu and Peng Liang [6] supported the Bagging algorithm for user review classification, finding that Bagging performed better than Naive Bayes and J48. Research by Arfiani and Rustam [2] showed that Bagging and Random Forest can classify ovarian cancer with 100% accuracy on Bagging. Indriana et al [7] compared Bagging and J48 to classify pathology on the Vertebral Column dataset, showing that the combination of Bagging and J48 increases accuracy. Raul et al's research [8] used Deep Learning with CNN models for requirements classification, while Tomas et al [9] and Urszula et al [10] focused on vector coding techniques such as word2vec and CBOW.

Overall, this research aims to compare the performance of Bagging algorithm with SVM for software requirements classification, and prove that the feature extraction technique of word2vec can provide good results.

Research Method

Dataset Normalization

The dataset utilized in the research is initially raw and unclean, prompting the need for text normalization. The dataset comprises three columns (ProjectID, RequirementText, and class). In the first step of normalization, the ProjectID column is removed, and techniques are applied to clean the data within the RequirementText column. The normalization methods employed are:

1. Tokenization: Assigning a value to each sentence in the data to facilitate entry into the vector space.
2. Stop word removal: Eliminating words deemed unimportant, such as "and, the, is, etc."
3. Changing uppercase letters to lowercase: Ensuring uniformity during machine learning training to optimize model performance.
4. Lemmatization: Transforming words with symbols, such as converting "running" to the base word "run".

These normalization techniques contribute to enhancing the data for optimal performance during the vectorization and training processes.

Vektorization Technique

In this stage, the normalized and tokenized dataset undergoes conversion into a numerical vector using the Neural Network vectorization technique, specifically word2vec. Two models, Continuous Bag of Words (CBOW) and Skip-gram, are employed in the research. Leveraging the word2vec model allows the capture of semantic and correlated words, ultimately enhancing the accuracy of the trained model [9] [11].

The CBOW architecture comprises an input word, a hidden layer or projection, and an output word. Its primary objective is to predict the output word when provided with input words surrounding the target word [9]. For instance, in the sentence "Ken lives in Kendal district," the word "lives" serves as the target to be predicted. To make this prediction, input words around the target are utilized.

Skipgram is the opposite of CBOW, consisting of word input, hidden layer/projection and word output. If CBOW uses n-hot encoded input with one-hot encoded output, then Skipgram uses one-hot encoded input with n-hot encoded output. The goal of Skipgram is to predict the context (output) around the current word (input) [9].

Model Training

In this training stage, using 2 algorithm models, namely Support Vector Machine and Bagging.. To avoid overfitting, this study uses crossvalidation. From the crossvalidation stage, the dataset will be divided into k folds, in this study using 10-fold to avoid overfitting, as in [1]. The dataset will be divided into 9 training subsets covering 90% of the total dataset and 1 subset will be the test set covering 10% of the total dataset [1].

Support Vector Machine

Support Vector Machine (SVM) is a widely utilized set of supervised learning machines known for tasks such as classification, regression, and outlier detection. In this research, SVM is applied to classify software requirements into multiple classes. Specifically, a linear kernel is employed in this study. SVM functions by testing and identifying a hyperplane that effectively separates two or more data classes in the given dataset.

Bagging

Bagging classifier is one of the ensemble learning techniques in machine learning. Basically Bagging a set of base Classifiers, will later aggregate their predictions to become the final prediction or averaged to produce a prediction value.

In this research, the sample dataset undergoes resampling to create specified bags. Following the resampling, each bag is individually trained using a machine learning classification model, specifically Support Vector Machine (SVM). The predictions from each model are then aggregated or averaged to obtain the final prediction value. Despite training the model three times with consistent preprocessing and model parameters, variations in the output were observed. Therefore, the results presented in this study are based on the outcomes from the last compilation.

Measuring Performance

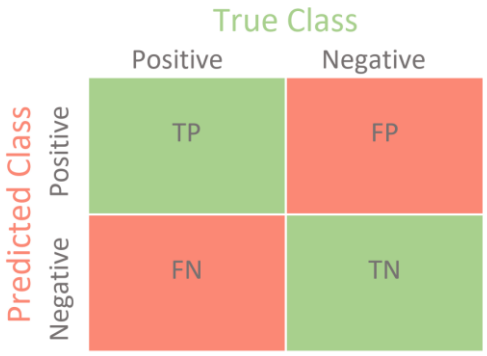

Figure 4 Confusion Matrix

In this study, model performance is assessed using evaluation metric techniques, including accuracy, precision, recall, and F1-score. The evaluation involves understanding the confusion matrix components:

TP (True Positive): Correctly predicted positive labels.

TN (True Negative): Correctly predicted negative labels.

FP (False Positive): Incorrectly predicted positive labels.

FN (False Negative): Incorrectly predicted negative labels.

Now, let's briefly discuss each metric:

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

A commonly used metric, calculated as the ratio of correctly predicted labels (True Positive + True Negative) divided by the total number of labels. However, it may be less reliable if the dataset is imbalanced.

Precision

$$Precision = \frac{TP}{TP + FP}$$

Precision measures the accuracy of positive label predictions. It focuses on the ratio of correctly predicted positive labels to the total predicted positive labels, helping to assess the model's ability to avoid false positives.

Recall

$$Recall = \frac{TP}{TP + FN}$$

Recall quantifies the accuracy of correctly predicted positive patterns. It considers the ratio of correctly predicted positive labels to all actual positive labels, providing insights into the model's capacity to identify positive instances.

F1-Score

$$F1 - Score = 2\frac{(Precision * Recall)}{(Precision + Recall)}$$

F1-Score represents the harmonic mean of Precision and Recall, serving as a balanced measure of overall model performance.

These metrics collectively provide a comprehensive evaluation of the model's effectiveness in classification tasks.

Analysis of Results

This research will analyze results by creating four models:

1. SVM algorithm with CBOW vectorization technique.
2. SVM algorithm with skip-gram vectorization techniques.
3. Bagging algorithm with CBOW vectorization techniques.
4. Bagging algorithm with skip-gram vectorization techniques.

These models will be trained and evaluated using precision, recall, and f1-score metrics. A comparison of the four models will determine which one exhibits the best performance based on the evaluation metrics.

Result and Discussion

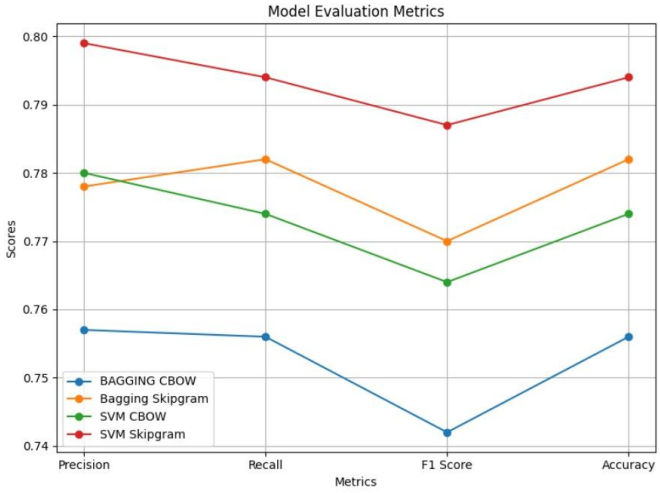

Figure 5 Result Classification with Multi Class

Table 1 Result Classification Multi Class with Training set & Testing set

	Training				Testing			
	Precision	***Recall***	***F1-Score***	***Accuracy***	***Precision***	***Recall***	***F1-Score***	***Accuracy***

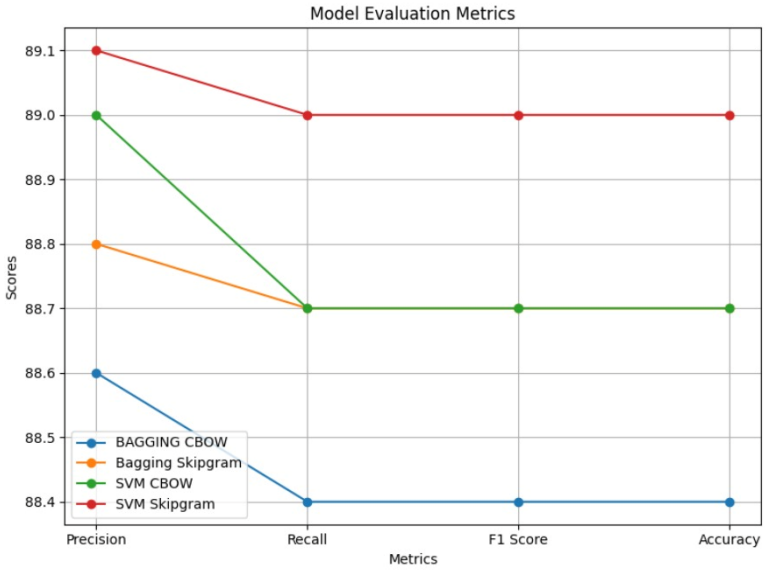

Figure 6 Result Classification with Binary Class

SVM CBOW	97.5	97.4	97.4	97.4	78	77.4	76.4	77.4
SVM Skipgram	96.9	96.7	96.7	96.7	79.9	79.4	78.7	79.4
Bagging CBOW	96	95.8	95.7	95.8	75.7	75.6	74.2	75.6
Bagging Skipgram	95.2	95.1	95	95.1	77.8	78.2	77	78.2

In Table 1 the testing data has a fairly far value vulnerability, this is called overfitting. This happens because of classifying software requirements in Multi Class, where there is no balance of classes in the training set and test set. What is meant by Multi Class is like Figure 5 that represents the amount of data and its classes. In the non-functional requirement class, it is translated into non-functional requirement sub-classes with an unbalanced amount of data between one class and the functional requirement class. Therefore, this research also classifies and analyzes for Binary Class.

Table 2 Result Classification Binary Class with Training set & Testing set

	Training				**Testing**			
	Precision	***Recall***	***F1-Score***	***Accuracy***	***Precision***	***Recall***	***F1-Score***	***Accuracy***
SVM CBOW	95.4	95.4	95.4	95.4	89	88.7	88.7	88.7
SVM Skipgram	95.2	95.2	95.2	95.2	89.6	89.2	89.2	89.2

Bagging CBOW	95.5	95.5	95.5	95.5	89.1	88.8	88.8	88.8
Bagging Skipgram	95.3	95.3	95.3	95.3	88.8	88.7	88.7	88.7

In comparing Table 2 and Table 1, overfitting is not a significant concern as the classification is based on a Binary Class (Functional Requirement and Non-functional Requirement). The inclusion of non-functional requirement data from the Multi-Class does not result in imbalance, as it has numerous sub-classes that are balanced with the functional requirement class. Table 2 indicates a 10% increase in all values for the test data compared to Table 1.

Table 1 shows a stable prediction value on the test data with a slight difference ranging from 1.2% to 1.6%. This difference is attributed to the number of sub-classes classified along with the amount of data/sampling in the training and testing sets during the cross-validation process. The stability in model performance in Table 2, as evidenced by consistent recall, accuracy, and f1-score values, suggests a balanced division of classes and samples in the training/testing sets, enabling the model to learn the data more accurately.

In this context, Skipgram proves more suitable for predicting words around its input compared to predicting empty words (CBOW). The hyperparameter tuning of word2vec can further enhance model accuracy. The conclusion drawn is that, in software requirements classification with the PROMISE_exp dataset, cleaned data exhibits rarity or variability from one data point to another. Skipgram's suitability for identifying infrequent, or rare, words is highlighted.

Furthermore, the research explores classification with an increased test dataset, ranging from 50 to 969. The findings indicate that as the dataset size increases, accuracy improves. This evaluation extends to both multi-class and binary class scenarios, providing insights into the model's accuracy across different methods.

Table 3 Result Classification Multi Class with Incrase Data

Model	***N data***	***Precision***	***Recall***	***F1-Score***	***Accuracy***
CBOW-SVM	50	32.4	31.9	30.1	31.9
	100	61.5	61.9	59.4	61.9
	250	64.5	60.3	59.9	60.3
	400	66.3	65.5	63.9	65.5
	550	73.3	73	71.5	73
	700	75.2	74.1	73	74.1
	850	76.2	74.7	74	74.7
	969	78	77.4	76.4	77.4

Skipgram-SVM	50	52.8	55.9	51.4	55.9
	100	53.9	48	48.6	48
	250	61.8	62.7	60	62.7
	400	68.1	68.7	67	68.7
	550	65.4	65.4	63.9	65.4
	700	69.5	69.2	68.1	69.2
	850	73.9	71.1	71	71.1
	969	79.9	79.4	78.7	79.4
CBOW-BAG-GING	50	-	-	-	-
	100	-	-	-	-
	250	-	-	-	-
	400	-	-	-	-
	550	-	-	-	-
	700	69.5	70.7	68.4	70.7
	850	73.5	74.2	72.4	74.2
	969	75.7	75.6	74.2	75.6
Skipgram-BAGGING	50	-	-	-	-
	100	-	-	-	-
	250	-	-	-	-
	400	-	-	-	-
	550	-	-	-	-
	700	68/9	68.5	66.7	68.5
	850	71.7	71.6	70.1	71.6
	969	77.8	78.2	77	78.2

It turns out that in the case of this research, the Bagging Algorithm is not able to classify with a small amount of data. Moreover, data with many labels such as the multiclass above.

Table 4 Result Classification Binary Class with Incrase Data

Model	***N data***	***Precision***	***Recall***	***F1-Score***	***Accuracy***
CBOW-SVM	50	72.8	72	68.2	72
	100	79.9	76	75.9	76
	250	83.9	82.8	82.9	82.8
	400	85.5	84.7	84.7	84.7

	550	86.4	86.3	86.3	86.3
	700	88.7	88.7	88.7	88.7
	850	88.4	88.4	88.4	88.3
	969	89	88.7	88.7	88.7
Skipgram-SVM	50	79	65.9	66.1	65.9
	100	76.7	72.9	73.2	72.9
	250	83.5	82.4	82.4	82.4
	400	87.5	87.2	87.2	87.2
	550	86.7	86.7	86.7	86.7
	700	88.8	88.5	88.5	88.5
	850	89.4	89.1	89.1	89.1
	969	89.6	89.2	89.2	89.2
CBOW-BAG-GING	50	74	70	68.2	70
	100	74.8	64	64.6	64
	250	81.5	81.1	81	81.1
	400	88.6	88.2	88.2	88.2
	550	88	87.4	87.4	87.4
	700	87	86.7	86.7	86.7
	850	88.8	88.6	88.6	88.6
	969	89.1	88.8	88.8	88.8
Skipgram-BAGGING	50	68.3	65.9	63.6	65.9
	100	77.9	74.9	74.8	74.9
	250	82.9	81.6	81.6	81.6
	400	85.2	84.5	84.4	84.5
	550	86	85.6	85.6	85.6
	700	88.1	87.5	87.6	87.5
	850	88.5	88.3	88.3	88.3
	969	88.8	88.7	88.7	88.7

Data balance is the biggest influence in this research case. In Multi Class classification on data 50-550 Bagging algorithm cannot work but in Binary Class classification on all data can classify. In the case of Binary Class when 100 data, the model has produced optimal results in the SVM model. And in this study proves that SVM is better than Bagging.

Table 1 shows the occurrence of overfitting in both models, this occurs because the diversity of sub-classes classified has uneven data.

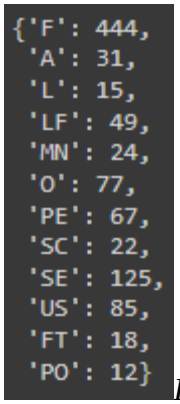

Figure 7 Jumlah Data Multi Class

Figure 7 shows the data imbalance between non-functional sub-classes. Take for example the SU or Security label, which only has 22 data. If you enter into cross-validation logic, with K-Fold 10, then from 969 data will be divided into 10. This means that 1 Fold may have more than 1 SU data or even none because the data is randomized. If the first class does not have SU data, it will affect the prediction value of the SU class. So in this research, it also examines binary classes or Functional and Non-functional requirements to prove that if there is no data imbalance between sub-classes, it will not experience overfitting.

Table 2 shows a decrease in overfitting, with a significant increase of 10%. This shows that the inequality in sub-classes in the data greatly affects the performance of the model.

This test uses the same hyperparameters between SVM models with CBOW vectorization techniques and Bagging models with Skipgram vectorization techniques, and so on. Tables 1 and 2 show that SVM is better than Bagging. SVM algorithm is 1.2% - 1.8% better than Bagging for Multi Class cases. In the case of binary class the difference is not far enough between Bagging and SVM algo, only 0.2% - 0.3% different. With Skipgram vectorization technique produces better accuracy than CBOW. That way problem formulation number 1 can be answered, that Bagging is not better than SVM in the classification of Software Requirement using word2vec vectorization preprocessing technique.

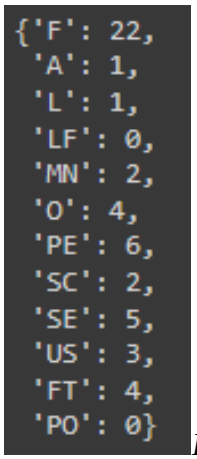

Figure 8 50 Datasets of Multi Class

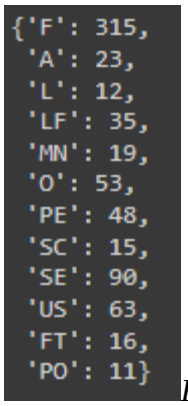

Figure 9 700 Datasets of Multi Class

To answer problem formulation number 2, it can be seen from tables 3 and 4 When viewed in table 3, the Bagging model cannot classify multi-class software requirements on a small dataset. Because of the inequality in classes on a small dataset, the bagging algorithm fails to classify.

Figure 8 above shows the imbalance of classes, this is what causes bagging to fail in classifying. However, table 1 shows that bagging can process when the data is 700 data of multi-class, as in Figure 9. In table 1 the Bagging Algorithm is able to classify with 50 data because this research wants to solve class imbalance by classifying in binary class. So actually Bagging can classify small datasets, but if there is a complex class division and results in class imbalance it will make it difficult for bagging to classify.

Conclusion

In conclusion, the SVM algorithm outperforms bagging in software requirements classification using CBOW and Skipgram vectorization techniques. This is attributed to the potential inconsistencies in values when compiling word2vec vectorization twice, leading to lower average values. Notably, bagging proves suitable for small datasets as well, not limited to large datasets, provided that the features or labels are not overly complex, and the data is balanced [6].

For future research, it is recommended to expand the PROMISE_exp dataset, particularly in the non-functional requirement sub-class. Additionally, optimizing hyperparameters for CBOW, Skipgram, and bagging algorithms can enhance model performance, ensuring the exploration of equivalent hyperparameters for more robust results. The current research employs original hyperparameters for the bagging algorithm, suggesting the potential for improvement with the discovery of optimal hyperparameters in future studies.

Daftar Pustaka

[1] E. Dias Canedo and B. Cordeiro Mendes, “Software Requirements Classification Using Machine Learning Algorithms,” *Entropy*, vol. 22, no. 9, p. 1057, Sep. 2020, doi: 10.3390/e22091057.
[2] A. Arfiani and Z. Rustam, “Ovarian cancer data classification using bagging and random forest,” presented at the PROCEEDINGS OF THE 4TH INTERNATIONAL SYMPOSIUM ON CURRENT PROGRESS IN MATHEMATICS AND SCIENCES (ISCPMS2018), Depok, Indonesia, 2019, p. 020046. doi: 10.1063/1.5132473.
[3] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, “What Works Better? A Study of Classifying Requirements,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Portugal: IEEE, Sep. 2017, pp. 496–501. doi: 10.1109/RE.2017.36.
[4] Z. Kurtanovic and W. Maalej, “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Portugal: IEEE, Sep. 2017, pp. 490–495. doi: 10.1109/RE.2017.82.
[5] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, “How can i improve my app? Classifying user reviews for software maintenance and evolution,” in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bremen, Germany: IEEE, Sep. 2015, pp. 281–290. doi: 10.1109/ICSM.2015.7332474.
[6] M. Lu and P. Liang, “Automatic Classification of Non-Functional Requirements from Augmented App User Reviews,” in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, Karlskrona Sweden: ACM, Jun. 2017, pp. 344–353. doi: 10.1145/3084226.3084241.
[7] I. Hidayah, E. P. Adhistya, and M. A. Kristy, “Application of J48 and bagging for classification of vertebral column pathologies,” in *Proceedings of the 6th International Conference on Information Technology and Multimedia*, Putrajaya: IEEE, Nov. 2014, pp. 314–317. doi: 10.1109/ICIMU.2014.7066651.
[8] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, “Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification,” in *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, Mérida: IEEE, Oct. 2017, pp. 116–120. doi: 10.1109/CONISOFT.2017.00021.
[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space.” arXiv, Sep. 06, 2013. Accessed: May 09, 2023. [Online]. Available: http://arxiv.org/abs/1301.3781
[10] U. Krzeszewska, A. Poniszewska-Marańda, and J. Ochelska-Mierzejewska, “Systematic Comparison of Vectorization Methods in Classification Context,” *Applied Sciences*, vol. 12, no. 10, p. 5119, May 2022, doi: 10.3390/app12105119.
[11] M. Ali Fauzi, “Word2Vec model for sentiment analysis of product reviews in Indonesian language”, in International Journal of Electrical and Computer Engineering (IJECE) Vol. 9, No. 1, February 2019, pp. 525~530. doi: 10.11591/ijece.v9i1.pp525-530